НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ

«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Інститут телекомунікаційних систем

Кафедра Інформаційно-телекомунікаційних мереж

До захисту допущено:

В.о. завідувача кафедри

_____ Лариса ГЛОБА

«___»_____2021 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційно-комунікаційні технології»

спеціальності 172 «Телекомунікації та радіотехніка»

на тему: «Алгоритм визначення оптимального часу надсилання сповіщень користувачу з використанням машинного навчання»

Виконав:
студент IV курсу, групи ТІ-72
Матвієнко Назар Тімурович _____

Керівник:
Доцент
Могильний Сергій Борисович _____

Рецензент:
с.н.с. НДІ телекомунікацій
Кайденко Микола Миколайович _____ _____

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____

Київ – 2021 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Інститут телекомунікаційних систем

Кафедра Інформаційно-телекомунікаційних мереж

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 172 «Телекомунікації та радіотехніка»

Освітньо-професійна програма «Інформаційно-комунікаційні технології»

ЗАТВЕРДЖУЮ
В.о.завідувача кафедри
_____ Лариса ГЛОБА
«___»_____2021 р.

ЗАВДАННЯ

на дипломну роботу студенту

Матвієнко Назару Тімуровичу

1. Тема роботи «Алгоритм визначення оптимального часу надсилання сповіщень користувачу з використанням машинного навчання», керівник роботи Могильний Сергій Борисович, доцент кафедри інформаційно-телекомунікаційних мереж ІТС, доцент, к.т.н., затверджені наказом по університету від «14» квітня 2021 р. № 1007-с
2. Термін подання студентом роботи 7 червня 2021 р.
3. Вихідні дані до роботи: наукові роботи про статистичний аналіз, машинне навчання, клас нейронних мереж LSTM, набір даних з eCommerce сервісу.
4. Зміст роботи:

   1. Провести аналіз існуючих методів передбачення часових рядів в статистичному аналізі та в машинному навчанні.

2. Проаналізувати принципи роботи LSTM класу рекурентних нейронних мереж.

3. Вибрати та обробити набір даних для навчання та тестування моделі.

4. Побудувати модель передбачення часових рядів LSTM класу для її подальшого застосування у відправці сповіщень. Проведення дослідження та застосування їх до покращення моделі LSTM.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):

   1. Титульний слайд;

   2. Актуальність;

   3. Основні задачі;

   4. Різниця між статистичним аналізом та машинним навчанням;

   5. Огляд існуючих методів статистичного аналізу;

   6. Огляд існуючих методів машинного навчання;

   7. Розгляд RNN та її підкласу LSTM мережі;

   8. Вибір та підготовка даних;

   9. Обробка даних з Python;

   10. Побудова моделі передбачення з LSTM;

   11. Отримання результатів;

   12. Загальні висновки.

6. Дата видачі завдання 4 жовтня 2020 року.

Календарний план

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітка
1	Отримання базових навичок в машинному навчанні	04.10.2020 – 01.02.2021	Виконано
2	Аналіз існуючих методів машинного навчання	02.02.2021 – 10.03.2021	Виконано
3	Аналіз існуючих методів статистичного аналізу	11.03.2021 – 29.03.2021	Виконано
4	Порівняння методів статистичного аналізу та машинного навчання	30.03.2021 – 10.04.2021	Виконано
5	Вибір даних та їх обробка	11.04.2021 – 25.04.2021	Виконано
6	Побудова LSTM моделі передбачення	26.04.2021 – 02.06.2021	Виконано
7	Оформлення дипломної роботи	27.05.2021 – 07.06.2021	Виконано

Студент Назар МАТВІЄНКО

Керівник Сергій МОГИЛЬНИЙ

РЕФЕРАТ

Робота містить 76 сторінок, 38 рисунків та було використано 11 джерел.

Актуальність: В даній роботі досліджується історія активності користувачів на наборі даних з E-Commerce магазину. Дані містять активність користувачів на сервісі і несуть інформаційну цінність, коли користувачі відвідують сервіс. Тим самим ці дані можуть бути використані для вивчення поведінки та режиму дня користувачів.

Вивчення активності дасть змогу передбачати наступну ймовірність входу користувача на сервіс, або вивчення його “вільного часу”. Велика кількість бізнес-процесів роблять аби підвищити кількість трафіку на їхньому сервісі. Тоді як влучне відправлення сповіщень або нагадувань буде доречне для кожного індивідуального користувача, а не відправка всім в різні години, не говорячи про часові зони.

Мета роботи: підвищити ефективність передбачення часових рядів за допомогою машинного навчання LSTM моделі з ціллю їх використання надсилання сповіщень користувач на базі його активності.

Ключові слова: машинне навчання, часові передбачення, активність користувачів, персоналізовані сповіщення, нейронна мережа, обробка даних.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ……..….....9
ВСТУП ……....9
РОЗДІЛ 1
АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПЕРЕДБАЧЕННЯ ЧАСОВИХ РЯДІВ В СТАТИСТИЧНОМУ АНАЛІЗІ ТА В МАШИННОМУ НАВЧАННІ..12
1.1. Різниця між статистичним аналізом та машинним навчанням ……12
1.2. Огляд існуючих методів передбачень в статистиці ……..……....14
1.2.1. Схована модель Маркова ……...……....14
1.2.1.1. Термінологія в схованій моделі Маркова ……14
1.2.1.2. Порівняння з схованої моделі Маркова з рекуретними нейронними мережами ……...…….....17
1.2.2. Точковий процес ……..…….....19
1.2.2.1. Система підрахунків N(t)…….……...……..…20
1.2.2.2. Умовна функція інтенсивності ($\lambda \cdot (t)$) ……...…20
1.2.3. Процес Хокеса ……...…...…21
1.2.3.1. Симуляції процесу Хокеса ……...…..……22
1.2.3.2. Слабкі сторони процесу Хокеса ……24
1.2.4. ARIMA ……...……...…...……26
1.2.4.1. Принцип роботи на прикладі ……...……...…...26
1.2.4.2. Авто-регресія …….……..……...27
1.2.4.3. Ковзна середня ……...……..……...28
1.2.4.4. Стаціонарність ……...……..……..29
1.2.4.5. Підсумок ARIMA ……..……...……..30
1.3. Основні принципи в машинному навчанні ……..……...……..31
1.3.1. Процедура машинного навчання ……...……...…….32
1.3.2. Нейрон як компонента машинного навчання ….....…….34
1.3.3. Функція активації …….……...……35

1.3.4. Активаційна функція ReLU ……...……37
1.4. Клас CNN …….……...……..38
1.4.1. Принцип роботи CNN ……..……...……39
1.4.2. Шар об’єднання (pooling) ……...……...……42
1.4.3. Підсумок CNN ……..……...…….43
1.5. Висновок……..……...……...44
РОЗДІЛ 2
АНАЛІЗ ПРИНЦИПІВ РОБОТИ LSTM КЛАСУ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ ……...…….……46
2.1. Клас RNN …….……..46
2.1.1. Принцип роботи RNN ……..……....…46
2.1.2. Часове зворотне поширення помилки …...……....48
2.1.3. Основні проблеми стандартних RNN ……...…….……....50
2.2. Нейронні мережі LSTM ……...……...50
2.2.1. Покроковий розгляд LSTM структури …….……...53
2.3. Висновок …….……...56
РОЗДІЛ 3
ВИБІР ТА ОБРОБКА НАБОРУ ДАНИХ ДЛЯ НАВЧАННЯ ТА ТЕСТУВАННЯ МОДЕЛІ …….……...……...……...…...57
3.1. Використані дані в роботі ……..……...……..58
3.2. Обробка даних з Python ……..…….…...……..59
3.3. Висновок ……...……65
РОЗДІЛ 4
ПОБУДОВА МОДЕЛІ ПЕРЕДБАЧЕННЯ ЧАСОВИХ РЯДІВ LSTM КЛАСУ ДЛЯ ЇЇ ПОДАЛЬШОГО ЗАСТОСУВАННЯ У ВІДПРАВЦІ СПОВІЩЕНЬ ……...…….……66
4.1. Розділення даних для навчання ……...……...66
4.2. Нормалізація даних ……...……...…….....68
4.3. Побудова моделі …….……..70

4.4. Отримання результатів та підсумок ……..……71
ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ……..…..…….……75
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ ……...…....76

ПЕРЕЛІК СКОРОЧЕНЬ

HMM	Hidden Markov Model
ARIMA	Auto Regressive Integrated Moving Average
LSTM	Long short-term memory
RNN	Recurrent neural networks
PPP	Poisson point process
MA	Moving average
AR	Autoregression
MSE	Mean squared error
ReLU	Rectified Linear Unit
CNN	Convolutional neural network
RGB	Red Green Blue
BPTT	Backpropagation through time
GRU	Gated recurrent units

ВСТУП

Актуальність: Часові передбачення – це методика або техніка для приблизного передбачення майбутніх подій. Передбачення часових рядів є чудовим інструментом, якщо певна бізнес-логіка або дія проявляє в собі закономірні патерни які очікуються виникнути ще раз. Для передбачення використовуються історичні дані з минулого, які є ключовими індикаторами для майбутніх подій.

В даній роботі досліджується історія активності користувачів на наборі даних з E-Commerce магазину. Дані містять активність користувачів на сервісі і несуть інформаційну цінність, коли користувачі відвідують сервіс. Тим самим ці дані можуть бути використані для вивчення поведінки та режиму дня користувачів.

Вивчення активності дасть змогу передбачати наступну ймовірність входу користувача на сервіс, або вивчення його “вільного часу”. Велика кількість бізнес-процесів роблять аби підвищити кількість трафіку на їхньому сервісі. Тоді як влучне відправлення сповіщень або нагадувань буде доречне для кожного індивідуального користувача, а не відправка всім в різні години, не говорячи про часові зони.

Безумовні переваги індивідуальних сповіщень користувачу:

- Залучення користувачів з одночасним підняттям трафіку сервісу (user engagement);
- Підвищення ефективності сповіщень сервісу завдяки персоналізації;
- Потужний інструмент для підвищення прибутковості, віддачі від окремого кожного користувача або інших показників сервісу;
- Широке застосування (e-commerce, новинні портали, навчання).

Мета й завдання дослідження

Мета роботи: підвищити ефективність передбачення часових рядів за допомогою машинного навчання LSTM моделі з ціллю їх використання надсилання сповіщень користувач на базі його активності. Для цього досягнення потрібно виконати **основні задачі:**

1. Аналіз існуючих методів передбачення часових рядів в статистичному аналізі та в машинному навчанні.

2. Аналіз принципів роботи LSTM класу рекурентних нейронних мереж.

3. Вибір та обробка набору даних для навчання та тестування моделі.

4. Побудова моделі передбачення часових рядів LSTM класу для її подальшого застосування у відправці сповіщень. Проведення дослідження та застосування їх до покращення моделі LSTM.

Об’єкт дослідження: передбачення, використовуючи LSTM моделі нейронних мереж.

Предмет дослідження: машинне навчання передбачення часових рядів.

Теоретичний результат дослідження: покращена модель передбачень часових рядів для оптимізації виділення часових ресурсів необхідних для обробки даних на одного користувача.

Практичний результат роботи: реалізація та тестування моделі передбачень з використанням LSTM моделі.

РОЗДІЛ 1

АНАЛІЗ ІСНУЮЧИХ МЕТОДІВ ПЕРЕДБАЧЕННЯ ЧАСОВИХ РЯДІВ В СТАТИСТИЧНОМУ АНАЛІЗІ ТА В МАШИННОМУ НАВЧАННІ

1.1 Різниця між статистичним аналізом та машинним навчанням

Передбачення часових рядів може відбуватися за допомогою статистичного аналізу так і машинного навчання. В свою чергу, варто підмітити, що машинне навчання є розвитком статистичного аналізу, так само як фізика є розвитком математики, щоб описати нові явища з використанням “знань” та інструментів. Машинне навчання вже в науковому суспільстві з 1959 року, проте тільки останнє десятиліття пропонує потужності техніки, що дозволяють їх ефективне використання.

Критично важлива точка розуміння різниці між статистичним аналізом та машинним навчанням.

Статистичний аналіз сильний та підтверджений часом як інструмент для виконання висновків, що отримуються за допомоги створення та насадження даних на конкретну (залежну від задачі) модель. Такі моделі дають змогу обрахувати чисельне значення, що певний ефект виник під впливом кокретних (передбачених моделлю) чинників, а не шуму. Проте ряд методів статистичного аналізу, як і машинного навчання можуть бути використані для аналізу висновків так і виконання передбачень. В роботі буде виконано огляд кількох важливих методів, як-от: Hawkes процес та схована модель Маркова.

В свою чергу машинне навчання концентрується на передбачення, використовуючи загальні алгоритми для знаходження патернів в громіздкому та негнучкому наборі даних.

Машинне навчання працює ефективно з даними широкого “розмаху”,

де кількість вхідних змінних перебільшує кількість сутностей які отримуються на виході, а не “довгі” дані, де кількість вхідних змінних менше кількості сутностей на виході.

Машинне навчання менш вразливе до менш обережного збирання даних та бути ефективними при нелінійних співвідношеннях даних. Проте попри ефективні передбачення машинного навчання, нестача ясності в моделях рішень може бути тяжко віднести до існуючих біологічних знань.

Класична статистика та машинне навчання відрізняються спроможністю осилити збільшення кількості змінних, що стосується досліджуваних сутностей. Статистичне моделювання було створено для обробки даних з кількома десятками вхідних змінних та малої або середньої кількості вибірок. В такій ситуації ця модель буде виконувати свою роботу ефективно, але як тільки число вхідних змінних та їх можливі співвідношення збільшується, тоді цій моделі складніше знайти ці залежності один між одними. Звідси, статистика починає бути менш точною та межа між нею та машинним навчанням стає нечіткою.

Ці два методи застосовується в залежності від поставленої задачі, машинне навчання більш стійке до негнучких даних та здатне до вивчення патернів і передбачення. Статистика більш спрямована на отримання висновків та знаходження співвідношення між змінними параметрами.[1] Проте обидва методи є слабкими відносно пояснення які саме параметри найбільше впливають на кінцевий висновок і по своїй природі більше націлені на результат. При цьому статистика вимагає вибору моделі на базі вже існуючих знань про систему, а машинне навчання вимагає вибору оптимального алгоритму передбачення, спираючись на його вузьконаправлену ефективність в конкретній задачі. В даних реаліях ефективність роботи моделі оцінюється шляхом оцінки чи результати є задовольняючі та адекватними для системи.

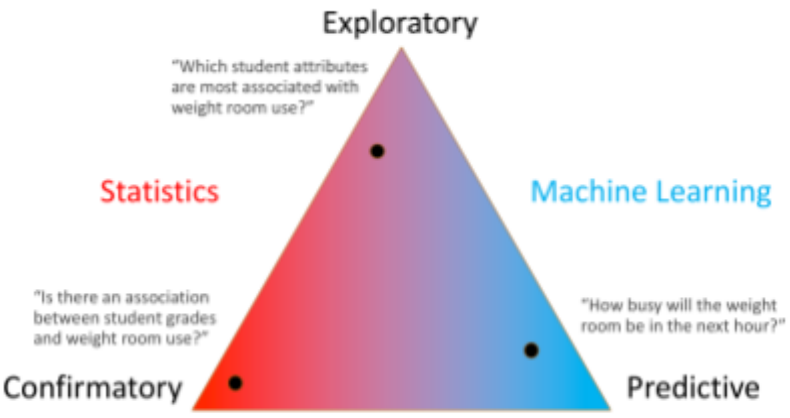

Рис. 1.1 Візуальне представлення можливостей машинного навчання та статистики

Попри все вище зазначено варто підчеркнути, що машинне навчання та статистичний аналіз є взаємодоповнюючими, оскільки після побудови моделі машинного навчання для аналізу її роботи зручно використовувати статистику для оцінки її роботи та редагування моделі на базі її результатів.

Підсумовуючи машинне навчання – це розділ науки про дані та аналітики, що веде до ефектиктої автоматизації та штучного інтелекту. Тоді як статистика є розділом математики, де було створено значна кількість інструментів та методик для передбачення.

1.2 Огляд існуючих методів передбачень в статистиці

Схована модель Маркова

Схована модель Маркова – це статистична модель Маркова тільки з умовою, що в системі, де відбувається моделювання припускається присутність схованих станів. Така модель ефективного використовується в розпізнаванні мови, жестів, розмічування частин мови, біоінформатиці та інших напрямках.

Термінологія в схованій моделі Маркова

Схований стан є першим компонентом в моделі Маркова відразу

після спостережень. Процес Маркова розглянуто на діаграмі 1.2, де “Дощ” та “Сонце” є схованими станами.

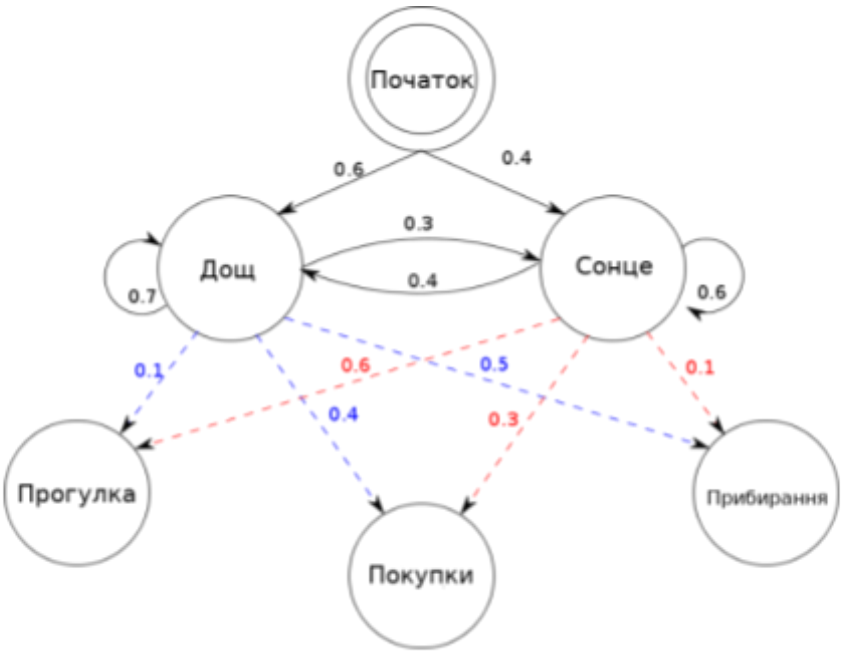

Рис. 1.2. Сховані стани, ймовірності переходів та спостереження

Спостереження – це дані про які вже зарані відомо та на рисунку 1.2 зображені як “Прогулка”, “Покупки”, “Прибирання”. Такі компоненти знаходять свої аналогії в машинному навчанні, спостереження – це дані на якій тренується нейронна мережа та сховані стани – гіпер параметри моделі, про які мова буде йти детальніше в наступних розділах.

Модель Маркова має ряд важливих параметрів:

T – довжина послідовності спостереження;

N – кількість станів в моделі;

M – кількість спостережень;

$Q = \{q_0, q_1, \ldots, q_{N-1}\}$ – стани в процесі Маркова;

$V = \{0, 1, \ldots, M - 1\}$ – можливий набір спостережень;

A – ймовірність переходів зі стану;

B – матриця ймовірностей спостережень;

π – початковий розподіл ймовірностей;

$O = (O_0, O_1, \ldots, O_{T-1})$ – початковий розподіл ймовірностей.

На рисунку 1.2 T поки немає спостережень, проте N = 2, M = 3, Q =

{“Дощ”, “Сонце”}, V = {“Прогулка”, “Покупки”, “Прибирання”}. Параметр A є стрілками, що вказують на кожен схований стан. **Матриця ймовірностей спостережень** B є синіми та червоними стрілками, що вказують на кожну з спостережень від схованих станів. По законам моделі Маркова всі ймовірності додаються і дають в сумі одиницю.

Звідси можна вивести наступні **ключові** рівності:

$$a_{ij} = P(\text{стан } q_j \text{ в } t + 1 \mid \text{стан } q_i \text{ в } t),$$

що означає, що ймовірність переходу в наступний стан q_j в момент часу t+1, за умови, що стан модель була в стані q_i в момент часу t.

$$b_j(k) = P(\text{спостереження } k \text{ в } t \mid \text{стан } q_i \text{ в } t),$$

що вказує, що ймовірність виконання спостереження k в момент часу t при тому що модель була в схованому стані q_i.

Початковий розподіл вказує на, те як моделі потрібно діяти в початковий момент і такий розподіл повинен бути зарані відомий. Повна модель з відомими ймовірностями переходів між станами, переходів від станів до спостережень та початковим розподілом ймовірностей позначається, як

$$\lambda = (A, B, \pi)$$

Проблеми які вирішує схована модель Маркова:

1. Якщо дана модель λ яка ймовірність виконання послідовності O ?

2. Якщо дана модель λ та послідовність спостережень O, яка послідовність схованих станів є найоптимальнішою ?

3. Якщо дана послідовність O спостережень та кількість схованих станів, яка найоптимальніша модель яка максимізує ймовірність

послідовності O ?

Приклад та вирішення до нього першої проблеми: ймовірність, що перше спостереження буде “Прогулка” буде рівна добутку ймовірності початкового розподілу на ймовірності переходів між станами за 2 кроки рівна $0.6 \cdot 0.1 + 0.4 \cdot 0.6 = 0.3$ (30%). Для другої проблеми прикладом задачі є: якщо дано модель та послідовність спостереження {“Покупки”, “Прибирання”, “Прогулка”} тоді найбільша ймовірність, що погода була {“Дощ”, “Дощ”, “Сонце”} з ймовірністю ~1.5%. Або дана модель з {“Прибирання”, “Прибирання”, “Прибирання”} скоріше за все погода була {“Дощ”, “Дощ”, “Дощ”} з ймовірністю 3.6%. Тому коли відбувається спостереження “Прогулка”, то найімовірніше погода не була “Дощ”.

Для останньої задачі прикладом є передбачення мови, в такому випадку збираються дані для навчання моделі, де модель знайде найоптимальніші ймовірності переходів між схованими станами та спостереженнями. По третій задачі, видно що машинне навчання та статистичний аналіз приходять один одному на допомогу щоб покращити результат роботи, або поєднуються для вирішення нових та складніших типів задач.[2]

Порівняння з схованої моделі Маркова з рекуретними нейронними мережами

Модель Маркова вважається простішої, ніж рекурентна нейронна мережа (RNN) і часто робить припущення, які не є дійсними в світі, проте якщо такі припущення є правдивими для певної системи, тоді ця модель може бути кращим вибором, оскільки RNN вимагає тонкого налаштування.

RNN може краще справитися з великим набором даних, оскільки додаткова складність може дати кращі результати, виділяючи більше інформації з даних. Ключове припущення схованої моделі Маркова:

Переходи між станами тільки залежать від поточного стану, а не на будь-які в минулому.

Таке припущення досить не часто є правдивим для більшості областей, де можна застосувати передбачення, знайшовши закономірність в даних. Наприклад, нехай ми хочемо передбачити кожну хвилину дня чи людина в стані **сну** чи **пробудження** на базі даних про її рух. Ймовірність того, що людина перейде з стану **сну** в стан **пробудження** збільшується з часом, тобто чим більше людина в стані **сну**. RNN модель була побудована для такого і теоретично здатна передбачити таке відношення, даючи вищу ймовірність передбачення. Можливим вирішенням цієї проблеми є включення попередніх результатів як ознаки, або визначення складених станів, проте додана складність не завжди покращує HMM точність передбачення і це точно не сприяє обчислювальним ресурсам.

Загальна кількість станів має бути відома зарані

Повертаючись до прикладу зі сном, поза двома цими станами важливі також інші стани, незважаючи на те, що нас цікавить лише стан сну та пробудження. Модель може отримати значну вигоду від додаткових станів, як от водіння, приймання душу і т.д., тобто події після яких зазвичай людина переходить в стан сну. Знову ж, RNN теоретично здатна вивчити таке співвідношення, якщо дати достатню кількість прикладів для навчання[3].

Вперше аналізуючи сховану модель Маркова виникає відчуття, що рекурентні нейронні мережі побудовані на базі цієї моделі і не мають новизни технології, проте це помилка і цю помилку можна пояснити в двох рівняннях:

$$p(x_1 \dots x_n) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_2) \dots,$$

що показує, що схована модель Маркова припускає, що ймовірність переходів між станами тільки залежить від поточного стану, а не на будь-які в минулому.

$$p(x_1 \ldots x_n) = p(x_1) \cdot p(x_2|x_1) \cdot p(x_3|x_2, x_1) \cdot p(x_4|x_3, x_2, x_1) \ldots,$$

рівняння показує, що якщо провести аналогію між RNN та HMM, RNN намагається та був створений з логікою, що ймовірність переходів між станами залежить не тільки від поточного стану, а ще й від минулих та цю залежність динамічно знаходиться.

Точковий процес

В сучасному світі існує багато подій коли при їх виникненні вони можуть бути бути збудником інших подій. Наприклад, ймовірність землетрусу зазвичай підвищує ймовірність землетрусів в регіоні де він відбувся. Таке підвищення ймовірності відбувається через збуджені землетрусом поштовхи плит. Іншим прикладом є панічна продаж акцій в одній країні може призвести до панічної продажі в іншій.

Проте крім збудження подія може бути ефектом пригнічення для наступного, наприклад, пожари в лісах. Пожари в лісах знижують ймовірність появи інших пожарів в наступних роках. Таке зменшення ймовірності зумовлене через горіння існуючого лісного палива, кількість якого зменшується після першого пожару. Звідси можна зробити висновок, що однакові події можуть бути збудниками для майбутніх послідовних схожих подій.

Якщо ймовірність схожих подій підвищується, тоді такий процес називається **стохастично збудженим** або **самозбудженим процесом**. Якщо подія має ефект пригнічення на наступні схожі події, як-от приклад з лісовими пожежами, такий процес називається **стохастично пригніченими** або **саморегульованими** процесами. Якщо ймовірність

подібного ефекту не впливає на схожі наступні події, то такі події можна описати за принципом **Poison Process Point** (точковий процес Пойзона), що розглянутий далі. Щоб зрозуміти принцип Хокеса необхідно дослідити принцип на якому він базується – точкові процеси. Проаналізуємо принцип роботи точкових процесів на часовому проміжку на якому відбуваються ряд подій.

Система підрахунків N(t)

Нехай маємо проміжок часу на якому є відмітки, які означають час виникнення події T_1, $T_2 \ldots T_N$, при цьому T_i вважається часом виникнення *i*-ої події. Така подія може бути землетрусом або лісовою пожежею. Щоб зробити модель таких подій замість використання T_1, $T_2 \ldots T_N$ використовують підрахунок N(t), який описує кількість виникнень події до часу t, нехай, якщо в Гімалаях за 80 років було зареєстровано 146 землетрусів, тоді N(80) = 146. Звідси, можна виділити історію виникнень події до часу u як H(u).

Умовна функція інтенсивності ($\lambda \cdot (t)$)

Аналізуючи концепції стохастичності доречно буде сформувати умовну функцію інтенсивності *($\lambda \cdot (t)$)*, яка характеризує очікування виникнення події в часі t, яка представлена нескінченно малим значенням виникнення події в навколо часу t. Рівність є умовною до попередньої історії H(t) точкових виникнень до часу t.

$$\lambda \cdot (t) = \lim_{h \to 0} \frac{E[N(t+h) - N(t) | H(t)]}{h},$$

рівняння умовної інтенсивності для опису часових точкових процесів N(t) за умови історії H(t).

Було розглянуто термін як самозбудження та саморегулювання, які можна зрозуміти, використовуючи функцію умовної інтенсивності. Якщо нещодавнє виникнення в історії H(t) збільшує значення функції умовної інтенсивності, тоді процес називається самозбуджуючим. Взагалі, $(\lambda \cdot (t))$ залежить не тільки від часу *t, а й від часу* T_i попередніх подій, тобто H(t).

Якщо система підрахунку N є **точковим процесом Пойзона**, тоді функція умовної інтенсивності $\lambda \cdot (t)$ залежить від поточного стану, але не на історії H(t). Точковий процес Пойзона не є самозбуджуючим і не є саморегльовним. В такому випадку $\lambda \cdot (t)$ є функцією опису сталого Пойзоновського точкового процесу з сталим значенням $\lambda \cdot (t) = \alpha$для всіх *t*, що означає, що ймовірність виникнення події є постійною в будь-який момент часу незалежно від частоти його виникнення до цього.

Процес Хокеса

Процес Хокеса належить до класу самозбуджуючих процесів названа на честь Алана Хокеса. Самозбуджуючі процеси використовуються в моделюванні подій, де відбувається тимчасове накладання подій один на одного. Такі події як “землетруси” та “продавання акції” часто вважається подіями з тимчасовим накладанням, тобто виникнення однієї події підвищує ймовірність виникнення в найближчому майбутньому. **Умовна функція інтенсивності процесу Хокеса** має вигляд:

$$\lambda \cdot (t) = \lambda + \sum_{t_i<t} \mu(t - t_i),$$

взявши, що $\{t_1, t_2 \ldots t_k\}$відомі спостереження виникнення подій до шуканого часу *t*,

λ −константа, яка означає інтенсивність фону,

μ −функція збудження, якщо вона рівна нулю, тоді такий процес

зводиться до спрощеного процесу Пойзона. Зазвичай, ця функція вибирається як експоненціальний розпад.

Функція експоненціального розпаду:

$$\mu(t) = \alpha \cdot e^{-\beta t},$$

α −сталий параметр, який можна описати як кожне виникнення події моментально підвищує ймовірність нового виникнення α.

β −сталий параметр, який описує спадання впливу події, на наступну подію.

Вплив зміни цих сталих параметрів є ключовим моментом при побудові моделі на базі процесу Хокесу та проаналізуємо симуляції в наступній частині.

Значним розширенням до самозбуджуючих процесів було впровадження **мульти-варіативності** або **взаємозбуджуючі точкові процеси**.

Маючи, $i = 1, 2, \ldots m$,тоді кожен процес підрахунку $N_i(t)$ має умовну інтенсивність

$$\lambda_i \cdot (t) = \lambda_i + \sum_{j=1}^{m} \sum_{t_{j,k}<t} \alpha_{i,j} e^{-\beta_{i,j}(t - t_{j,jk})}$$

Симуляції процесу Хокеса

Проаналізуємо симуляції процесу Хокеса з різними параметрами: константа, яка означає інтенсивність фонуλ, інтенсивність надходженняα, ступінь спаду β.

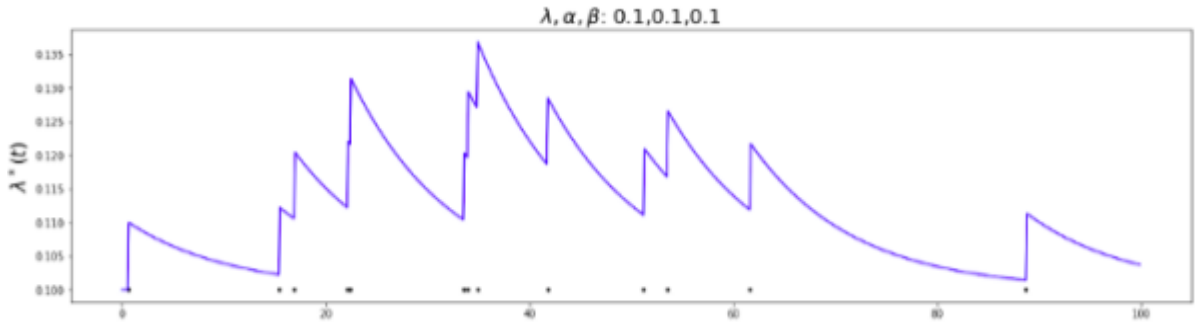

Рис. 1.3. Графік при значеннях 0.1

На графіку (рисунок 1.3) чітко видно збудження та експоненціальне спадання графіку. Видно, що чим ближче в часі одна подія слідує попередній події, тим більший ефект накладання та вплив попередніх. Збільшимо значення фонової інтенсивності λ до 0.5.

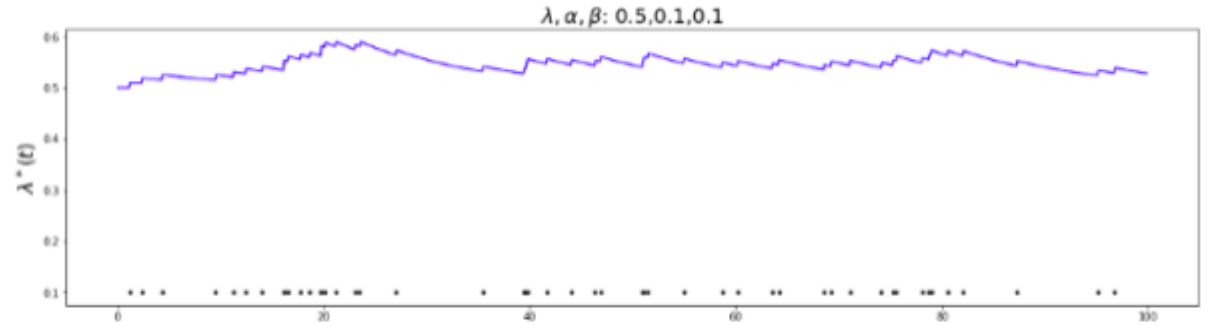

Рис. 1.4. Збільшення інтенсивності фону до 0.5

З графіку (рис 1.4) видно, що число подій збільшується зі збільшенням фонової інтенсивності та тримається значення вище 0.5. Розглянемо вплив констант інтенсивності та інтенсивності спаду на графіку. Збільшимоαдо 0.5.

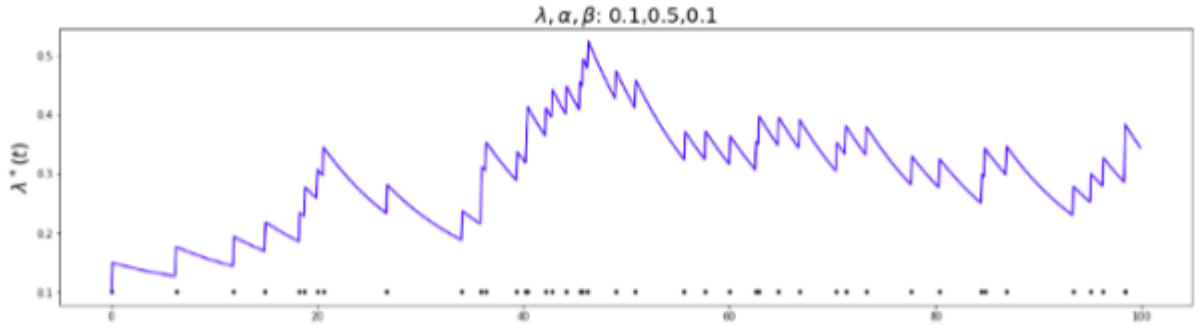

Рис. 1.5. Збільшення інтенсивності надходження до 0.5

З графіку (рисунок 1.5) видно, що кількість подій збільшується через те, що виконання події збільшує інтенсивність надходження нової події на значенняα.При цьому $\lambda \cdot (t)$збільшується швидше та лежить в межах від 0.1 до 0.6. Збільшимо інтенсивність спадуβдо 0.5, пам'ятаючи що ця константа контролює інтенсивність спадання події на всі послідовні події.

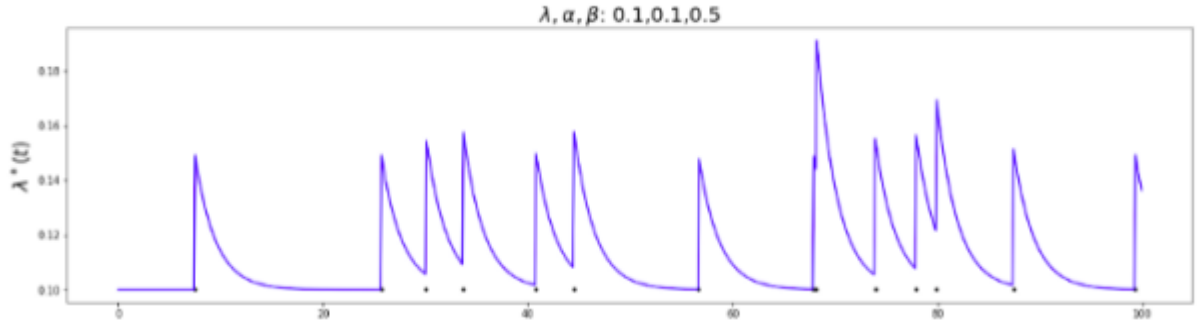

Рис. 1.6. Збільшення інтенсивності спаду до 0.5

Порівнюючи графіки з рисунків 1.3 та 1.6, можна побачити значно менший вплив спаду на умовну функцію інтенсивності$\lambda \cdot (t)$. Ключові концепції процесу Хокеса були приведені в даній симуляції.[4]

Слабкі сторони процесу Хокеса

Підсумовуючи основний принцип роботи процесу Хокеса, функція умовної інтенсивності базується на значеннях фонової інтенсивності та збуджуючої функції. Проте в реальних сучасних застосуваннях, гнучкість класичного процесу Хокеса є обмеженою через приближеність та

неточність в багатьох секціях поставленої задачі. В класичному процесі ми припускаємо, що кожне виконання події в системі збуджує майбутні події позитивно та акумулятивно (накладаючись) в природі та експоненціально спадаючи з часом. Параметри α, βрозуміються так, що кожне виконання події моментально збільшує інтенсивність виконання наαта спадає за певним законом. Параметрαне може захопити ефект, коли відбувається навпаки приглушення ефекту, що сильно обмежує модель та робить її не універсальною.

Виникає потреба в кращій моделі з меншою кількістю припущень, з такою ціллю було процес Хокеса запропоновано поєднання нейронною мережею яка б навчилася інтенсивностям події, що знову доказує, що статистичний аналіз та машинне навчання є **взаємодоповнюючими**.

Ефект минулих подій може бути адитивним так і віднімаючим, а також може впливати на майбутні події в залежності від порядку минулих подій. Вважається, що все ж не просто точно передбачити землетруси та їх складні слідуючі наслідки з експоненціальним законом спадання функції умовної інтенсивності. Тому можливо використати LSTM (Long Short-Term Memory), яка є покращеною версією рекурентних нейронних мереж, яка має проблему зникаючого градієнту, про яку буде детальна в наступних розділах. LSTM оброблю повну послідовність даних замість одиночних точок даних. Нейрона умовна функція інтенсивності дозволяє динамічно навчитися на базі історії минулих подій H(t) важливих параметрів як-от: число, порядок та час виникнення минулих подій. Нейронний процес Хокеса відрізняється від експоненціального спадаючого процесу Хокеса, вирішуючи наступні проблеми:

1. Фонова інтенсивністьλне є постійною, а є змінною з часом;
2. Збудження є не монотонним, тому що вплив при мульти-варіативності на $\lambda \cdot (t)$може бути збуджуючим або пригнічуючим;
3. Модель може бути узагальнити вплив один події на інші події,

також враховуючи що одні пари подій не мають зовсім ефекту на один одного.

Підсумовуючи, класичний процес Хокеса є доволі вузьконаправленим інструментом для передбачення подій, проте поєднання з LSTM дає змогу обійти цю проблему і дає здатність виконувати задачі іншого складнішого напрямку, де є не тільки збудження, але й пригнічення подій іншими послідовностями подій[5].

ARIMA

ARIMA, яка означає інтегрована модель авторегресії ковзного середнього – є інструментом з області статистичного аналізу, який використовується в аналізі часових рядів для кращого їх розуміння, або для передбачення майбутніх подій.

Принцип роботи на прикладі

Нехай поставлена задача передбачити ціну бензину в конкретний день – неділю. Ціна в неділю буде залежати від ціни в суботу і нехай добавимо залежність ціни від п’ятниці і опишемо їх:

S_t – ціна за бензин в неділю

S_{t-1} – ціна за бензин в суботу

S_t – ціна за бензин в п'ятницю

Тоді ціна в неділю залежить від ціни в суботу (2), яка в свою чергу залежна від ціни в п’ятницю (1), а також тут можливий непрямий перехідний вплив між ціною в п’ятницю та неділі (3).

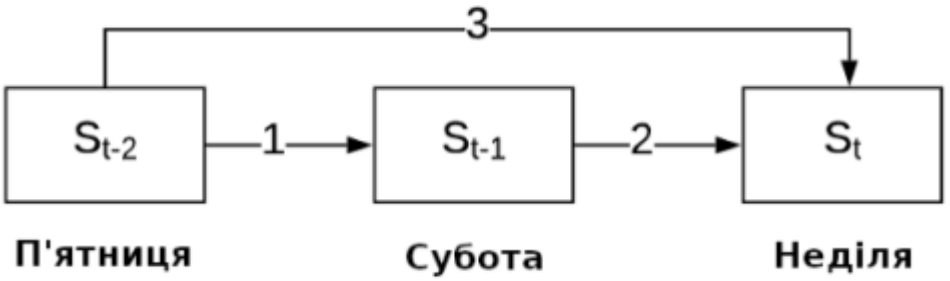

Рис. 1.7. Можливі впливи на ціну

Такий зв’язок між цінами називається **кореляцією**, якщо значення обох змінних збільшується (або зменшується) разом, тоді вони є позитивно корельованими. Якщо значення однієї збільшується, а іншої зменшується (або навпаки), тоді вони є негативно корельованими.

Кореляція поділяється на два типи:

● Автокореляція, яка враховує прямі та непрямі ефекти (1, 2, 3 на рисунок 1.7);

● Часткова автокореляція, яка враховує тільки прямі ефекти (2, 3 на рис 1.7).

Звідси можна визначити фактор відставання, цей фактор рівний 2-м в даному випадку, оскільки береться два попередніх дні.

Авто-регресія

В авто-регресії поточні значення є корельованими зі значеннями попередніх значень і тип кореляції тут є частковою автокореляцією. Рівняння для авто-регресії виглядає наступно:

$$Y_t = \beta_1 + \Phi_1 Y_{t-1} + \Phi_2 Y_{t-2} + \ldots + \Phi_p Y_{t-p},$$

відповідні коефіцієнти/ваги $(\Phi_1, \Phi_2 \ldots \Phi_p)$відносяться до відповідних попередніх спостережень і визначаються кореляцією між поточним

спостереженням та відповідним попереднім спостереженням ($Y_{t-1}, Y_{t-2}... Y_{t-p}$).

Варто підмітити про **(p)** який називається порядок відставання, який означає кількість попередніх спостережень (відставань) ми включаємо в модель, тобто кількість відставань які мають значну кореляцію між поточним спостереженням.

Ковзна середня

Ковзна середня (далі MA) модель аналізує наскільки неправильно було передбачено попередні значення для попередніх часових проміжків з ціллю отримати кращі апроксимовані результати в поточному часовому періоді. Ефект цих попередніх помилок спостережень (відставань) впливають на поточний спостереження шляхом автокореляції між цими помилками.

Рівняння середньої ковзної:

$$Y_t = \beta_2 + \omega_1 \varepsilon_{t_{-1}} + \omega_2 \varepsilon_{t_{-2}} + ... + \omega_q \varepsilon_{t_{-q}} + \varepsilon_t \ ,$$

змінна ε представляє собою помилки у відповідні відставання та їхні коефіцієнти/ваги ($\omega_1, \omega_2... \omega_p$)які розраховані статистично в залежності від кореляції.

Знову ж, варто підмітити про **(q)**, що представляє собою розмір вікна, що рухається, тобто кількість помилок відставань (спостережень) які мають сильний вплив на поточне спостереження. Ця змінна схожа на (p), але враховує помилки замість власне спостережень. Таким чином MA модель допомагає AR моделі враховувати помилки попередніх спостережень підвищуючи точність поточного.

При комбінації AR та MA рівнянь отримаємо ARMA:

$$Y_t = (\beta_1 + \beta_2) + (\Phi_1 Y_{t-1} + \ldots + \Phi_p Y_{t-p}) + (\omega_1 \varepsilon_{t-1} + \ldots + \omega_q \varepsilon_{t-q} + \varepsilon_t)$$

Стаціонарність

Стаціонарність даних – це вимога для ефективної роботи ARIMA для роботи з часовими рядами. Для часових ряд бути стаціонарними потрібно задовольняти 3 умови:

1. Середнє значення (μ)повинне бути константне;
2. Стандартна девіація(μ)повинна бути константна;
3. Сезонність не повинна існувати в послідовності.

Нехай отримано числову послідовність з не константним середнім значенням. Чітко видно на рисунку 1.8, що середнє значення збільшується з часом, тобто послідовність не стаціонарною.

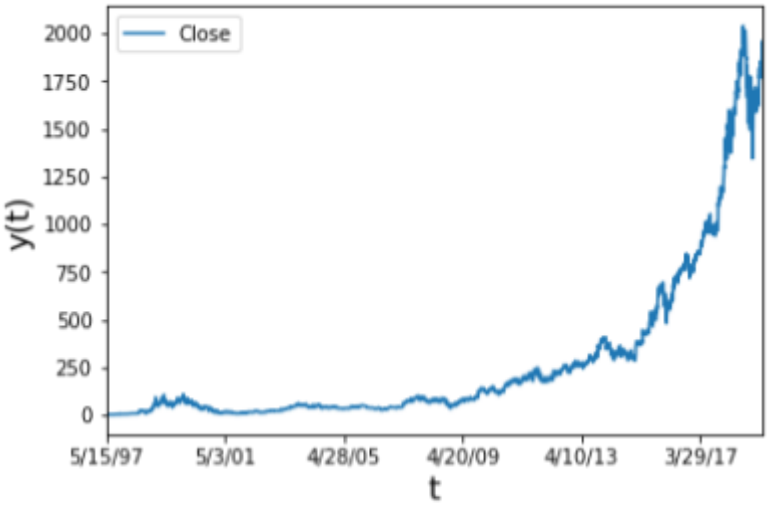

Рис. 1.8. Приклад графіку зі зростаючим середнім значенням

Задача прибрати цей зростаючий тренд шляхом знаходження різниці між послідовними часовими кроками, що еквівалентно:

$$Z_t = Y_{t+1} - Y_t$$

Після обрахунку значень отримаємо постійне середнє значення,

постійну стандартну девіацію та відсутність сезонності. В такому вигляді часова послідовність є **стаціонарною**.

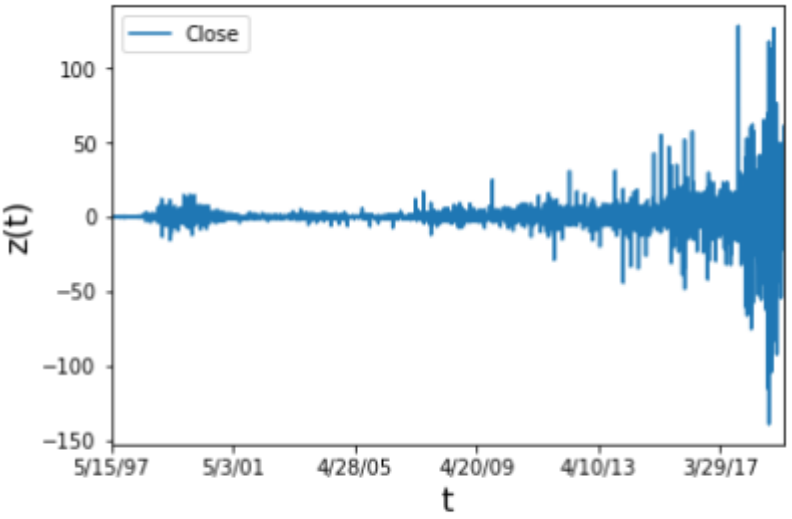

Рис. 1.9. Приклад стаціонарних значень

В ARIMA **(I)** означає інтегрованість і означає замість передбачення власне часових проміжків відбувається передбачення різниці послідовності покроково. Процедура різниці була виконана лише один раз, проте її можна робити декілька разів до тих пір поки дані будуть вважатися стаціонарними.

$$Z_t = Y_{t+1} - Y_t \quad => \quad d = 1$$

$$Q_t = Z_{t+1} - Z_t \quad => \quad d = 2$$

Така послідовність різниць є важливим параметром ARIMA і є визначним в успішності моделі. Модифікацією ARIMA є SARIMA яка здатна працювати з даними сезонного типу, проте вимагає додаткових параметрів для налаштування[6].

Підсумок ARIMA

ARIMA зарекомендувала себе як простий та ефективний інструмент

для передбачення часових рядів. ARIMA є ефективною для однокрокових та декілька крокових однозмінних передбачень та є спрощеним конкурентом LSTM і може видавати кращі результати для коротких термінів, ніж LSTM.

1.3 Основні принципи в машинному навчанні

Машинне навчання – це ряд геометричних задач, розглянемо одну з них на прикладі лінійної регресії.

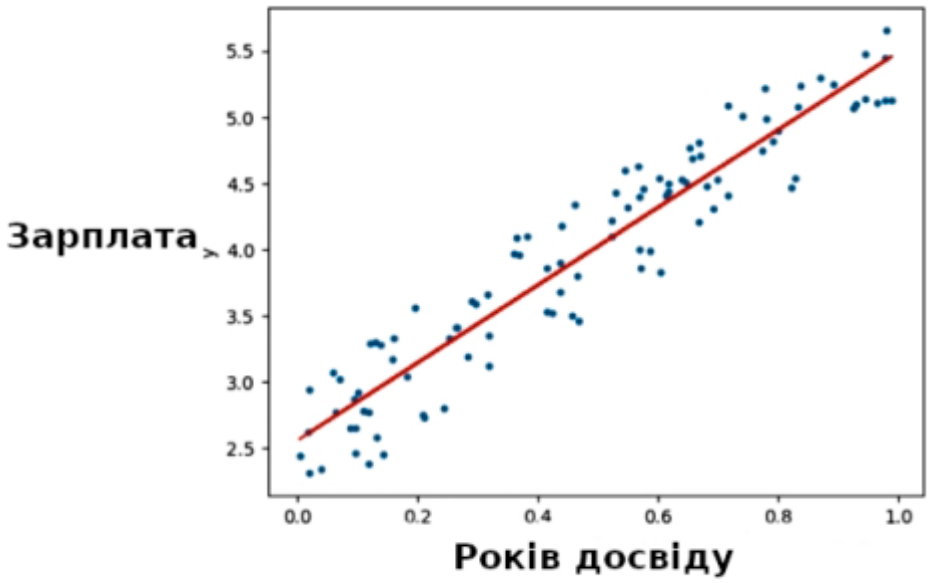

Рис. 1.10. Приклад лінійної регресії

Маючи задачу передбачити зарплату в залежності від досвіду роботи, потрібно отримати набір даних та побудувати пряму, яка проходить через ці точки.

Рівняння прямої:

$$y = mx + b,$$

де у випадку прикладу, у – знайдена зарплата,

х – заданий вхідний параметр, досвід роботи.

Щоб знайти цю пряму, в яку можна підставити вхідні значення та отримати вихідні значення, необхідно використати функцію помилок або функція вартості, або як її ще називають – функцією втрат. Розглянемо на прикладі часто використаній середньоквадратичної помилки формула якої має наступний вигляд:

$$MSE = \frac{1}{N} \cdot \sum_{i=1}^{N} (y_i - \widehat{y_i})^2,$$

де $\widehat{y_i}$передбачення для вхідний данихx_i,

різниця $(y_i - \widehat{y_i})$ показана на рисунку 1.11, що показує значення помилки передбаченого значення до цільового значення y_i.

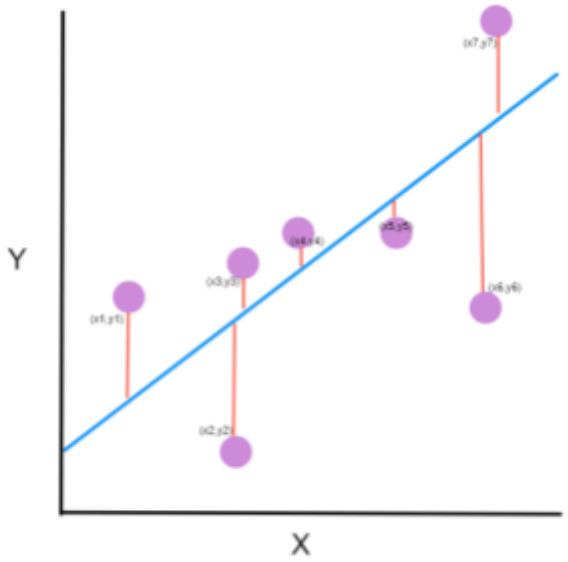

Рис. 1.11. Відхилення від лінійної регресії

Далі відбувається сумування цих помилкових значень і знаходження середнього значення. Варто підмітити про квадратичність, оскільки для ефективного аналізу та навчання моделі необхідно мати додатні числа.

Процедура машинного навчання

Мета машинного навчання з прикладу рисунку 1.10 є знаходження найоптимальнішого місця положення прямої на площині між точками. Це

можна зробити аналізуючи MSE значення та підбирання значення m – кут нахилу прямої та b – здвиг прямої в рівнянні $y = mx + b$.

Процес знаходження найоптимальнішого положенням прямої є мінімізація середньоквадратичного значення MSE, за допомогою **градієнтного спуску**.

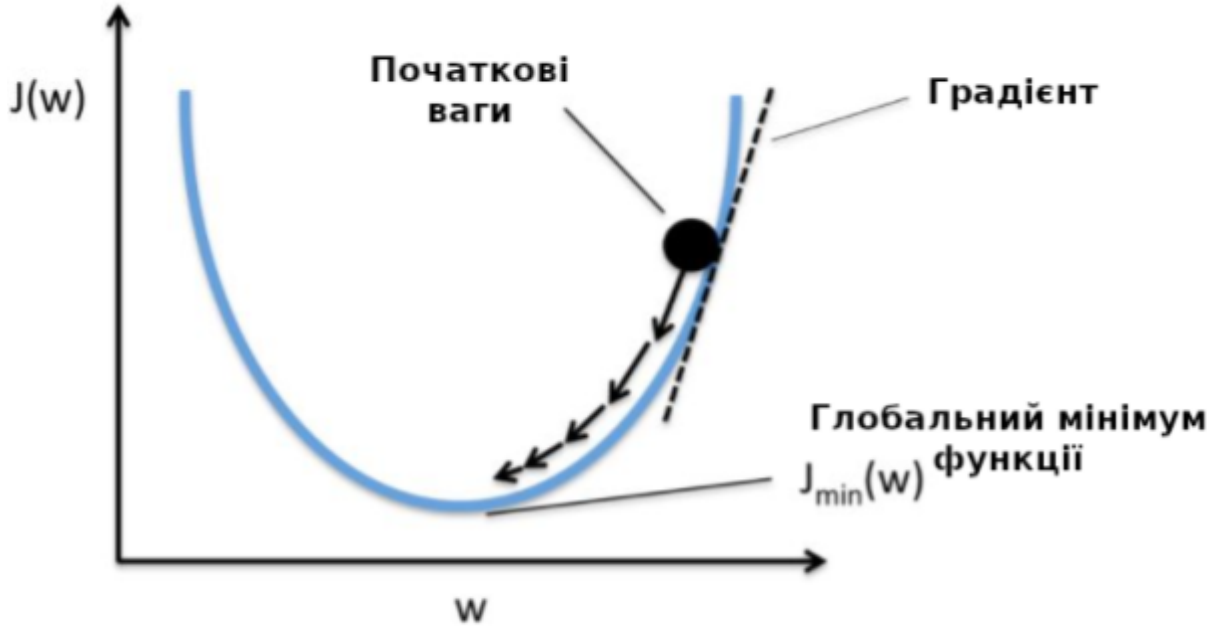

Рис. 1.12. Градієнтний спуск

Щоб знайти мінімум або максимум функції потрібно знайти точку, де похідна буде рівною 0 шляхом виконання повторюваного алгоритму знаходження локальних мінімумів. Починається пошук початкових випадкових вагів (рисунок 1.12) та рухаємося в напрямку негативного градієнту (площина < 0) і далі ітеративно продовжується рух в напрямку негативному градієнту в новій точці до тих пір поки не доберемося до глобальному мінімуму функції (рисунок 1.12). Розмір цих кроків називається темп навчання (learning rate). З високим значенням темпом навчання покривається більша відстань з кожним кроком, але з підвищеним ризиком промахнутися з точкою мінімуму, оскільки значення куту площини постійно змінюється. З надто малим значенням темпом навчання ми точно можемо впевнено можемо дістатися до точки мінімуму, оскільки

відбувається перерахування частіше. Низький рівень темпу навчання є точнішим, проте вирахування градієнту займає часові ресурси і це може зайняти не оптимальну кількість часу, що добратися до мінімуму функції вартості.

Повертаючись до лінійної регресії, кількість особливостей як “досвід роботи”, можуть бути декілька, або декілька десятків. Тоді рівняння регресії отримує розширений вигляд:

$$\hat{y} = w_1 x_1 + w_2 x_2 + b,$$

де x_1– досвід роботи, x_2– середня зарплата в сфері,

w_1– вага вхідного параметру x_1,w_2– вага вхідного параметру x_2,b– зміщення.

Коли особливостей x1-2 їх зручно записувати в рівнянні розписуючи, проте коли їх декілька десятків – їх розписування не є раціональним, тому з допомогою математичних перетворень було прийдено до зручного запису рівняння з багатьма вхідними сутностями x:

$$y = \sigma(\omega^T x + b) = \sigma(\sum_{d=1}^{D} \omega_d x_d + b),$$

при x – вектор вхідних сутностей та ω– вектор, який містить ваги відповідних вхідних сутностей. Тому відбувається виконання скалярного добутку між цими векторами, що значно дає змогу раціонально записувати активаційну функцію.

Нейрон як компонента машинного навчання

Коли градієнтний спуск є методом визначення навчених параметрів, сам компонент який приймає ці навчені параметри та пропускає через себе вхідні дані називається **нейроном**. Існує багато дебатів на рахунок вірності цього твердження чи варто його так називати, проте для простоти та через

аналогічність деяких базових функцій більшість схиляється до поняття нейрона. Одна з основних функцій нейрона є пропускання одних електричних сигналів та придушення інших в залежності від ваги дендритів (кількість синаптичних контактів). В штучному нейроні працює такий же принцип: відбувається сумування всіх вхідних параметрів з вхідних даних, або вихідні дані попередніх нейронів. Горизонтальна глибина з кількості нейронів більше 2-х, вважається глибоким навчанням.

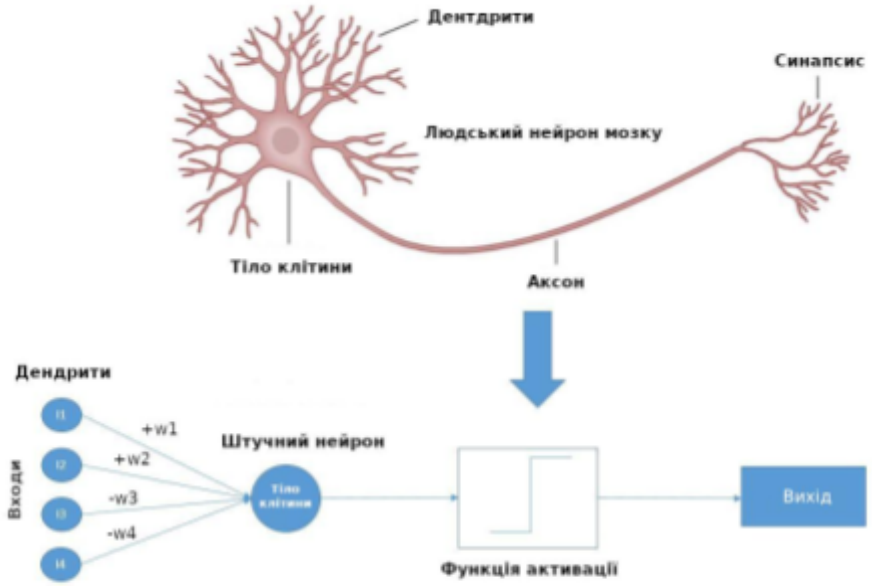

Рис. 1.13. Аналогія з нейроном

На рисунку 1.13 зображено додатні та від’ємні ваги, оскільки відбудеться сумування всіх вхідних значень, враховуючи їх ваги, та якщо ця сума буде більшої, ніж межа активації саме тоді на виході з’явиться сигнал, або вихідне значення в штучному нейроні. Якщо значення менше, ніж межа активації, тоді нічого не буде відбуватися.

Отже, кожен x_i є вхідним значенням, w_i є наскільки сильний вплив робить і-ий значення на вихідне значення та знак w_i вказує чи вплив є збуджуючим або пригнічуючим. Сума всіх вхідних значень та їх вагів додається до упередженості (або межа активації) **b** та передається сигмоїдної функції вказуючи чи активація відбулася (1) або ні (0).

Функція активації

Функція активації додана в нейронну мережу з ціллю допомогти мережі вивчити складні патерни в даних. Порівнюючи з нейроном такі функції є фінальною стадією вирішення що саме передається до наступного нейрону шляхом перетворення в певну форму та передавання далі по мережі.

Причини чому необхідно мати функції активації:

1. Необхідність значення в певних межах, оскільки неконтрольовані ваги *w*можуть досягати великих значень у глибоких мережах, де вивчаються мільйони параметрів, що неодмінно призведе до обчислювальних обмежень.

2. Найважливіша властивість функцій активацій – це можливість додавання **нелінійності** до нейронної мережі.

Поняття нелінійності є дійсно ключовим моментом в навчання мережі, оскільки ефект в початкових значеннях може давати великий вплив, тоді як зростання цих значеннях сповільнюється нелінійно та за деякою межею може не давати ефекту зовсім. Прости прикладом є віджимання людиною від підлоги, при поступовому підвищенні кількості віджимань м’язи людини стають витривалішими та сильнішими швидше, ніж після прогресування до 40 віджимань.

Вступною та більше навчальною функцією в простих задачах є сигмоїдна функція, формула якої виглядає наступно:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

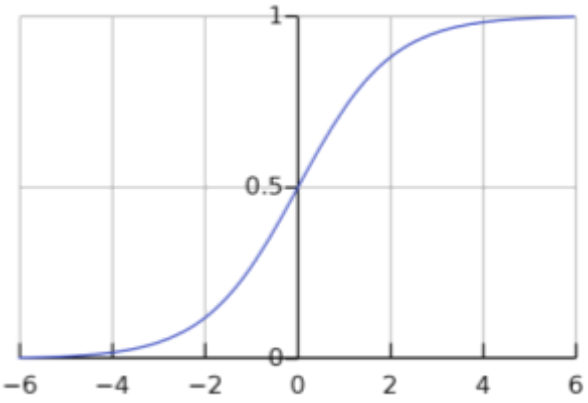

Рис. 1.14. Сигмоїдна функція

Існує ряд рішень які застосовуються як функції активації та вирішують слабкі сторони один одного, проте популярні функції мають хоча б одну з наступних проблем:

1. **Градієнтна проблема зникнення:** нейронні мережі навчаються, використовуючи градієнтний спад. Градієнтний спад складається із зворотного поширення, що є ланцюг дій для зміни вагів вхідних даних з ціллю зменшити помилку після кожного проходу через мережу(epoch). Градієнт – це набір похідних його складових, тому при проходженні через велику кількість шарів виникає проблема зникаючих значень та їх градієнтів та активаційна функція прямує їх нуля.

2. **Нуль-симетричні:** вихідні значення активаційної функції маю бути симетричними відносно нуля, щоб градієнт не здвигався в конкретному напрямку.

3. **Обраховувальні витрати:** активаційна функція застосовуються до кожного шару і повинні бути вирахувані мільйони разів в глибоких мережах, тому вони повинні бути обчислювально не важкими для виконання.

4. **Диференційована:** як зазначено, нейронні мережі навчаються за допомогою градієнтного спуску, тому шари мають бути диференційованими або частково диференційованими. **Це необхідна умова для функції бути функцією активації**[7].

Активаційна функція ReLU

ReLU запропонувала себе кращим засобом, ніж сигмоїдна функція, яка значно програє через зникаючий градієнт. ReLU пропонує більшу швидкість виконання через відсутність складної математики, що дає швидше навчання моделі. Також ReLU добирається до точки мінімуму вартості швидше, оскільки її лінійність гарантує відсутність плато, або насичення. Також ReLU пропонує розсіяність, що часто ефект який переслідуються при побудові, оскільки моделі намагаються імітувати біологічні людські нейрони, яких їх мільярди в тілі людини і вони не активовуються всі одночасно, а лише від залежності вхідних даних. Розсіяність пропонує більшу точність, даючи кращі передбачення та менший прояв перенавчання. Наприклад, нейрон який призначений для ідентифікації вух не повинен активовуватися при вхідному зображені на якому будівля.

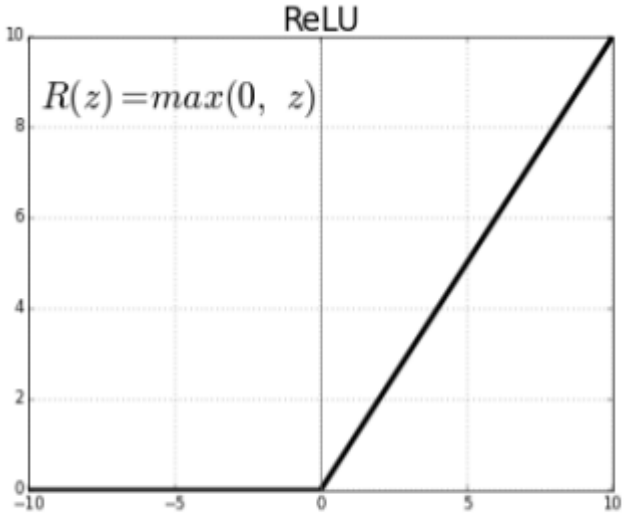

Рис. 1.15. ReLU функція

Специфічна проблема для найбільш популярної ReLU функції: ReLU не є нуль відцентровою тому страждає від **“вмираючого ReLU”**, оскільки вихідні значення рівні нуль при від’ємних значень, що призводить

деякі вузли нейронної мережі до повної деактивації, не вивчаючи нічого і в результаті можливий варіант, коли значна частина мережі не виконує нічого.

1.4 Клас CNN

CNN – це згорткова нейронна мережа, яка є класом машинного навчання, часто використана для аналізу медіа ресурсів. Зазвичай, нейронним мережам притаманні матричні операції множення, проте не у випадку CNN, який використовує спеціальну техніку яка називається Згортка. В математиці згортка – це математична операція двох функції, яка дає третю функцію, показуючи як розмір однієї функції змінився іншою.

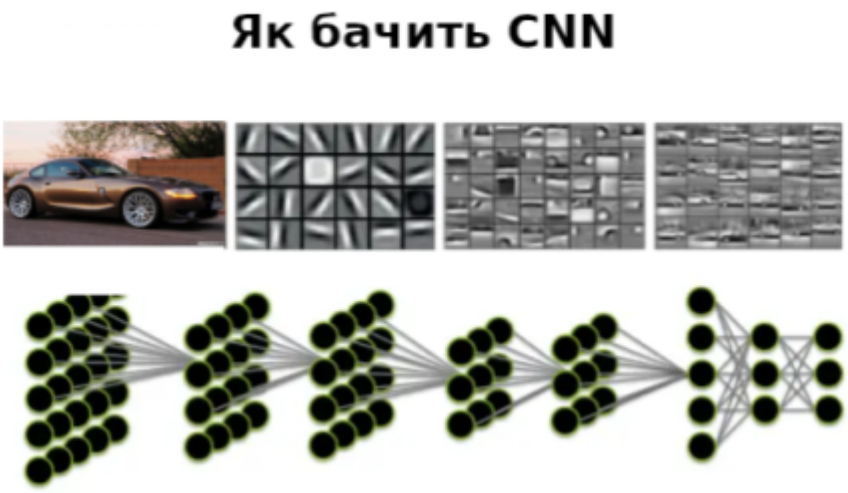

Рисунок 1.15. Розкладання зображення в CNN

Головний принцип CNN: завдання нейронної мережі є зменшити зображення у форми, які простіше обробити, без втрати особливостей (ліній), які є критичними для вдачного передбачення.

Принцип роботи CNN

Перед тим як проаналізувати CNN роботу, варто розглянути як представлені зображення. RGB зображення – це двовимірна матриця пікселів, на другому рівні якої знаходяться 3 значення кольорів, тоді

чорно-біле зображення на другому рівні має тільки 1 значення кольорів, тобто одновимірна матриця. Представлення RGB зображення на рисунку 1.16.

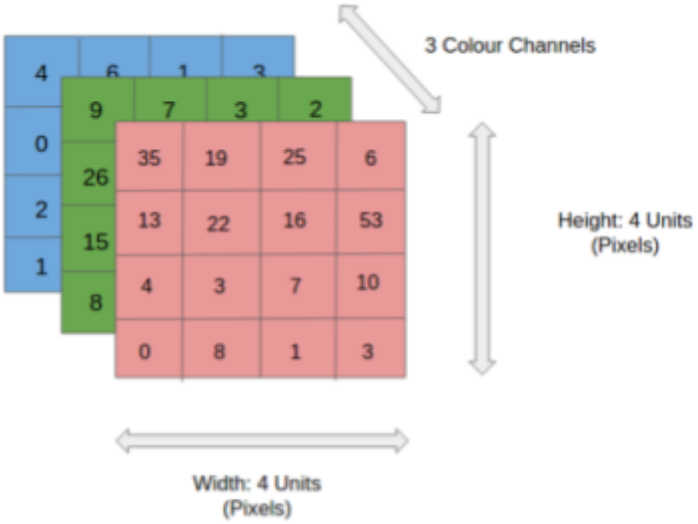

Рис. 1.16. Матричне представлення зображення

Розглянемо згортку на спрощеній ситуації – чорно-біле зображення. Першим кроком вибирається ядро (kernel). Ядро представляє собою одновимірною матрицею, наприклад, 3x3. Ядро є матричним представленням сутності (лінії, кола). Наприклад, нижче зазначені матриці відповідають за ядра, які знаходять вертикальні та горизонтальні лінії.

$$\begin{vmatrix} 0 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 1 & 0 \end{vmatrix} \qquad \begin{vmatrix} 0 & 0 & 0 \\ 1 & 1 & 1 \\ 0 & 0 & 0 \end{vmatrix}$$

Рис. 1.17. Матриця вертикальної та горизонтальної лінії

Матриця такого виду буде знаходити на зображенні лінію, що відокремлює два однакових кольори.

$$\begin{vmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{vmatrix}$$

Розглянемо основний принцип роботи CNN – згортку.

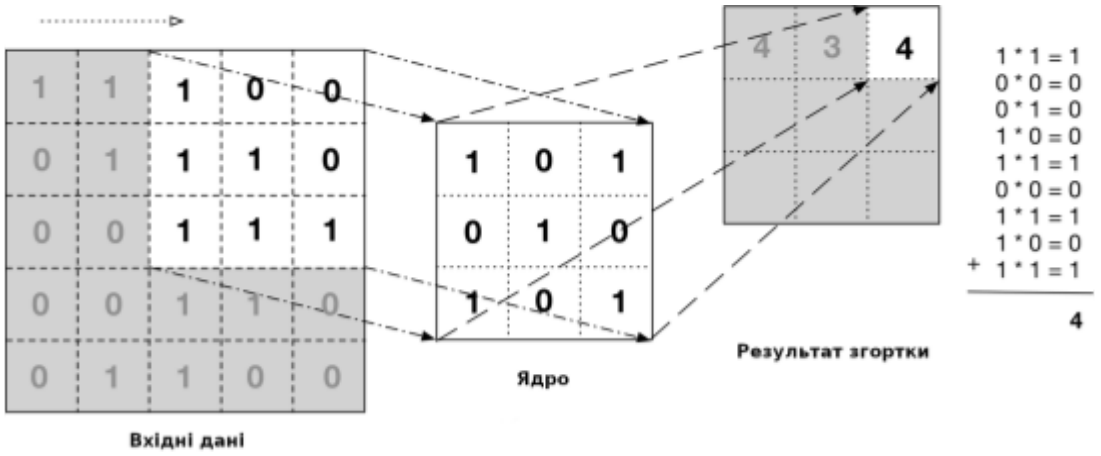

Рис. 1.18. Процедура згортки

На рисунку 1.18 вхідною точкою є **вхідні дані**, яка реальним представленням зображенням в матричному вигляді NxM, де N – висота зображення, M - ширина. Другим компонентом є ядро розміром K. Під час згортки наступні вхідні дані на інші шари з нейронів дані будуть мати вигляд:

$$D = 1 + (W - K + 2 \cdot P) / S,$$

де D – розмір висоти та ширини наступного зображення,

W – розмір ядра, S – здвиг руху матриці ядра.

P – розмір відступу,

Розмір відступу **(P)** використовується у випадку, коли зображення не поміщається повністю у вхідне зображення, тому існує два варіанти:

- Зробити відступи заповнені нулями, щоб ядро підходило;
- Відкинути частину зображення, де фільтр (ядро) не поміщається,

залишивши тільки правдиву частину зображення.

Здвиг руху матриці ядра **(S)** – це кількість пікселів, які здвигаються по поверхні вхідної матриці. Коли здвиг рівний 1, то рух відбувається 1 піксель одночасно, при здвигу рівному 2, рух відбувається 2 пікселі і т.д.

Згорткова нейронна мережа складається з множини шарів з штучних нейронів. Ці штучні нейрони є спрощеною імітацією біологічний аналогічних частин та є математичними функціями, які вираховують суму вхідних значень, враховуючи їх ваги та виводячи вихідне активоване значення. У випадку вхідних даних у форматі зображення, кожен шар генерує декілька активаційних функцій які передані в наступний шар.

Перший шар, зазвичай, знаходить базові сутності та особливості як горизонтальні та діагональні лінії. Ці вихідні дані далі передані в наступний шар який знаходить більш складніші особливості як кути, або комбінація кутів. Передаючи вихідні дані з шару в шар, мережа поступово навчається знаходячи більш складні особливості як об’єкти, обличчя і т.д.

На базі кінцевої активаційної карти останнього згорткового шару, класифікаційний шар виводить масив значень (від 0 до 1), який визначає ймовірність зображення належати до якогось класу. Наприклад, якщо CNN побудована для класифікацію котів, собак та коней, тоді останній шар буде представляти ймовірності появи одного з класів на вхідному зображенні.

Шар об’єднання (pooling)

Застосування CNN є ресурсоємним завданням, оскільки сучасна CNN мережа VGG здатна працювати із кольоровими зображеннями 224x224 при цьому мережі буде необхідно вивчити мільйони параметрів через присутність 150 528 особливостей.

Для значного прискорення навчання використовують шар об’єднання, який відповідає за зменшення розміру вже згорткованої сутності. Такий

шар дозволяє знизити кількість обраховувальний операцій, щоб опрацювати дані, зменшуючи розміри. Є два типи об’єднання: об’єднання по середньому числу та по максимальному значенні.

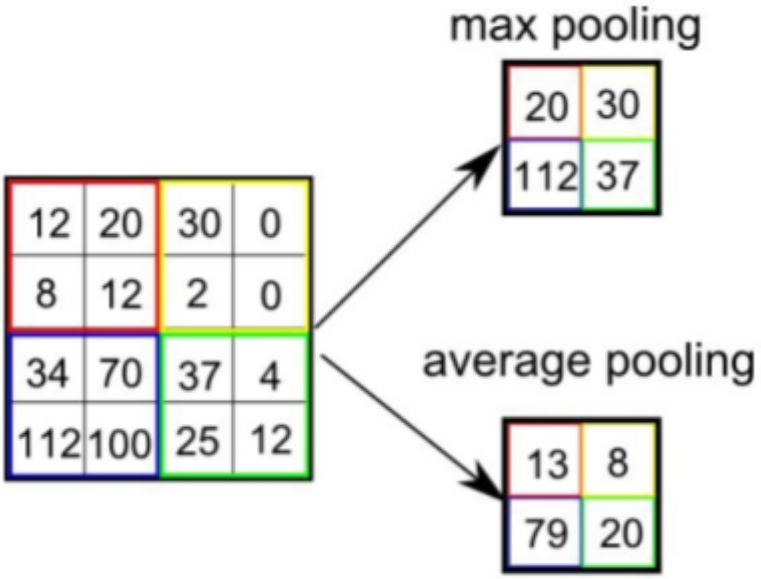

Рис. 1.19. Приклад шару об’єднання

В об’єднанні по максимальному значенні знаходиться максимальне значення з всієї вхідного матриці в рамках розміру ядра. Таким чином, Max Pooling виконує свою роботу і як подавляч шумів, оскільки він не пропускає шумні активовані значення. В той же час Average Pooling повертає середнє значення частини вхідної матриці в рамках розміру ядра, забезпечуючи зменшення розміру та придушування шумів[8].

Підсумовуючи даний шар, об’єднання по середньому значенні робить зображення більш гладким і приглушує більш контрастні особливості, які можуть бути не виявлені при цьому методі. Об’єднання по максимальному значенні вибирає яскраві та контрастні пікселі з зображення, що є корисним, де фон зображення є темним та задача поставлена на визначення світліших пікселів вхідних даних.

Підсумок CNN

При тяжких обчисленнях та ресурсах які CNN займає, така мережа

здатна видавати цінну інформацію. Головним принцип роботи мережі є знаходження патернів та деталей, які людина часто може не помітити. Проте спільна проблема класичної CNN мережі – це не знаходження контенту, **розуміння** цього контенту, тобто зв’язати речі на зображеннях та прийти до конкретного висновку.

CNN традиційно застосовуються для обробки зображень, така мережа здатна працювати з часовими рядами, проте не дає такі ж результати, ніж рекурентні нейронні мережі. Варто зазначити, що процедура згортки (визначення як форма однієї функції змінює форму іншої функції) є ефективним інструментом для комбінування з LSTM, даючи більш точніші передбачення. Таке комбінування можливе через використання розширених згорток – що дозволяє розрахувати розширення між кожними штучними нейронами, тобто розмір між ними, що дозволить нейронній мережі зрозуміти відношення між різними спостереженнями в часових рядах.

Висновок

Статистичний аналіз часто використаний для отримання висновків, що отримуються за допомоги створення та насадження даних на конкретну (залежну від задачі) модель. Такі моделі дають змогу обрахувати чисельне значення, що ефекти виникли під впливом деяких інших (передбачених моделлю) чинників, а не шуму чи випадковості. Проте ряд методів статистичного аналізу, як і машинного навчання можуть бути використані для аналізу висновків так і виконання передбачень.

В свою чергу машинне навчання концентрується на передбачення, використовуючи загальні алгоритми для знаходження патернів в громіздкому та негнучкому наборі даних.

Рекурентні мережі здатні краще справитися з великим набором даних, оскільки додаткова складність може дати кращі результати, виділяючи більше інформації з даних. Ключове припущення схованої моделі Маркова:

Переходи між станами тільки залежать від поточного стану, а не на будь-які в минулому. А також обмеженість в припущенні загальна кількість станів має бути відома зарані.

Підсумовуючи основний принцип роботи процесу Хокеса, функція умовної інтенсивності базується на значеннях фонової інтенсивності та збуджуючої функції. Проте в реальних сучасних застосуваннях, гнучкість класичного процесу Хокеса є обмеженою через приближеність та неточність в багатьох секціях поставленої задачі. В класичному процесі ми припускаємо, що кожне виконання події в системі збуджує майбутні події позитивно та акумулятивно (накладаючись) в природі та експоненціально спадаючи з часом.

ARIMA – простий та базовий, проте ефективний інструмент для передбачення часових рядів. ARIMA є ефективною для однокрокових та декілька крокових однозмінних передбачень та є спрощеним конкурентом LSTM і може видавати кращі результати для коротких термінів, ніж LSTM.

При тяжких обчисленнях та ресурсах які CNN займає, така мережа здатна видавати цінну інформацію. Головним принцип роботи мережі є знаходження патернів та деталей, які людина часто може не помітити. Проте спільна проблема класичної CNN мережі – це не знаходження контенту, розуміння цього контенту, тобто зв’язати речі на зображеннях та прийти до конкретного висновку.

CNN традиційно застосовуються для обробки зображень, така мережа здатна працювати з часовими рядами, проте не дає такі ж результати, ніж рекурентні нейронні мережі.

РОЗДІЛ 2

АНАЛІЗ ПРИНЦИПІВ РОБОТИ LSTM КЛАСУ РЕКУРЕНТНИХ НЕЙРОННИХ МЕРЕЖ

2.1 Клас RNN

Рекурентні нейронні мережі (RNN) є класом нейронних мереж, які є основним алгоритмічним представником для послідовних даних, або часових рядів. RNN використовується в голосовому помічнику Siri та у голосовому пошуку Google. Це перша нейронна мережа, яка пам’ятає свої вхідні дані у внутрішній пам’яті, що відіграє ключову та невід’ємну роль у вирішенні проблем, що пов’язані з послідовними даними. Дана мережа є інструментом деяких з найбільших досягнень у глибокому навчання за останні декілька років.

Які і більшість нейронних мереж, вона вже відома відносно довго, вперше вони були створені в 1980-х. В зв’язку з сильним зростанням кількості даних, які треба обробити та виділення більше ресурсів на їх обробку дало можливість розкрити потенціал RNN та винахід LSTM значно підросла їх застосування.

Через присутність внутрішньої пам’яті, RNN мережі можуть пам’ятати важливі речі про вхідні дані, дозволяючи їм бути набагато точнішими при передбаченні наступних подій. Тому RNN віддається перевагу як алгоритму для послідовних даних як часових рядів, розпізнавання мови, тексту, фінансові дані, аудіо- та відеоряди, погода і в багатьох інших сферах. Рекурентні нейронні мережі можуть формувати в рази глибше розуміння про послідовність та її контекст, порівнюючи з іншими алгоритмами.

Принцип роботи RNN

Для більш ефективного аналізу буде порівняно з нейронною мережею прямого поширення з використанням послідовних даних. Послідовні дані – це впорядковані дані, в яких співзалежні дані слідують один за одним, наприклад, порядок в ДНК. Найпопулярнішим типом послідовних даних вважаються часові ряди, що є послідовністю даних на проміжку часу.

Рекурентні та прямого поширення нейронні мережі отримали назви від їхнього типу поширення інформації. У мережах прямого поширенння інформація передається в одному напрямку – з вхідного шару, через сховані шари до вихідного шару. Інформація рухається прямо в мережі та ніколи не повертається у вузол (нейрон) двічі.

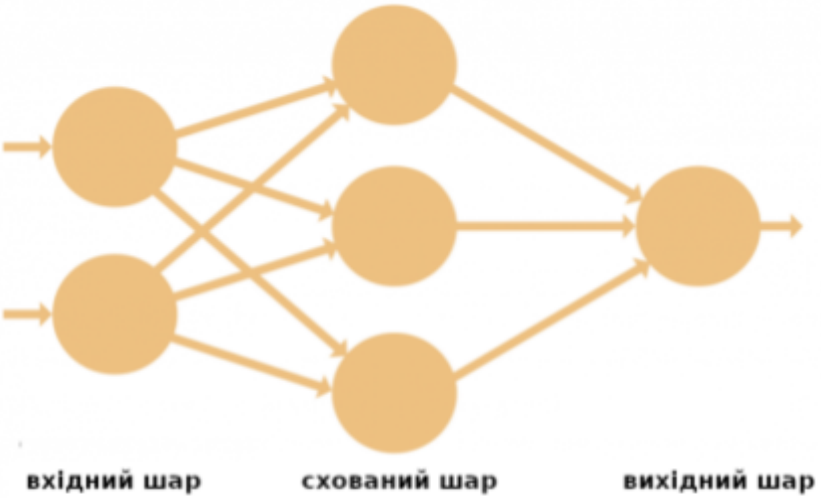

Рис. 2.1. Нейронна мережа прямого поширення

Мережі прямого поширення не мають пам’яті вхідних даних, тому погано передбачають наступний крок, оскільки вона враховує тільки поточні вхідні дані без єдиною підказки про вплив інших даних на часовому проміжку. Така мережа не здатна запам’ятовувати про минуле, окрім власне натренованих параметрів.

В мережі RNN інформація повертається назад за допомогою циклу. При наступному прийнятті рішення, вона врахує поточні вхідні дані, а

також попередньо вивчені дані. Класична RNN має короткочасну пам’ять, при комбінуванні з LSTM вони можуть мати довготривалу пам’ять.

Проаналізуємо приклад роботи обох цих мереж на прикладі. Нехай дано звичайну мережу прямого поширення зі словом “нейрон” як вхідні дані та обробка відбувається символ за символом. Коли мережа досягає літери “р”, мережа вже забула про літери “н”, “е”, та “й”, що означає неможливість також мережі передбачити який наступний символ буде наступним. Рекурентна нейронна мережа матиме можливість зберегти в пам’яті ці символи. Вона видасть вихідні дані, скопіює їх та перевідправить назад в нейрон.

Отже, RNN має два типи вхідних даних: поточні та нещодавні минулі, що важливо оскільки послідовність даних містить невід’ємну інформацію, яка впливає на те, що відбудеться в майбутньому, завдяки чому RNN здатний вирішувати задачі, що інші алгоритми не в змозі. Мережа з послідовним поширенням призначає матрицю ваги до вхідних даних та видає вихідні дані, при цьому RNN шукає матрицю як для поточних вхідних даних, так і минулих. А також, рекурентна нейронна мережа також змінює ваги для обох даних завдяки **градієнтному спаду** та **методу часового зворотного поширення помилки**.

Часове зворотне поширення помилки

В нейронних мережах відбувається пряме поширення помилки, щоб отримати вихідні значення моделі та перевірити ці дані на коректність, отримавши помилку. Зворотне поширення помилки повертається назад в нейронну мережу, щоб знайти часткові похідні помилки відносно вагів, що дає змогу відняти це значення від вагів. Ці похідні використовуються градієнтним спуском, що ітеративно мінімізує дану функцію, змінюючи ваги збільшуючи або зменшуючи в залежності від зменшення помилки, що є основним принцип навчання нейронної мережі.

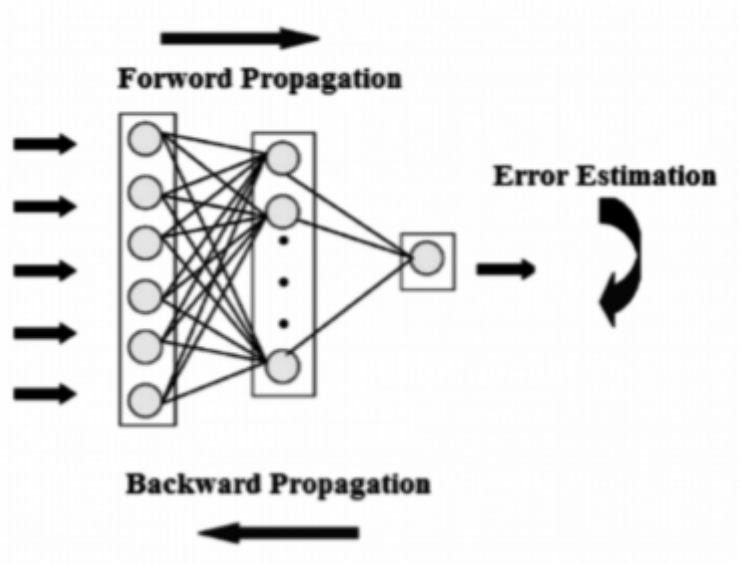

Рис. 2.2. Зворотне та пряме поширення помилки

Часове зворотне поширення помилки є модифікацією для виконання поширення помилки для розгорнутої RNN. Розвертання – це інструмент для візуалізації, який допомагає зрозуміти принцип роботи в мережі. Зазвичай, в більшості фреймворків програмування зворотне поширення помилки забезпечується автоматично. На рисунок 2.4 зображено розгорнуту RNN після знаку дорівнює. Варто підмітити, що циклічність відсутня, оскільки різні часові кроки візуалізовані та інформація просувається покроково, а також зображення показує, що RNN можна розглядати як послідовність нейронних мереж. В рамках BPTT помилка зворотно поширюється від останнього до першого часового кроку, при розгортці всіх часових кроків, що дає змогу розраховувати помилку для кожного кроку, оновлючи значення вагів. Потрібно взяти до уваги, що BPTT є обчислювально дорогою процедурою при великій кількості часових кроків.

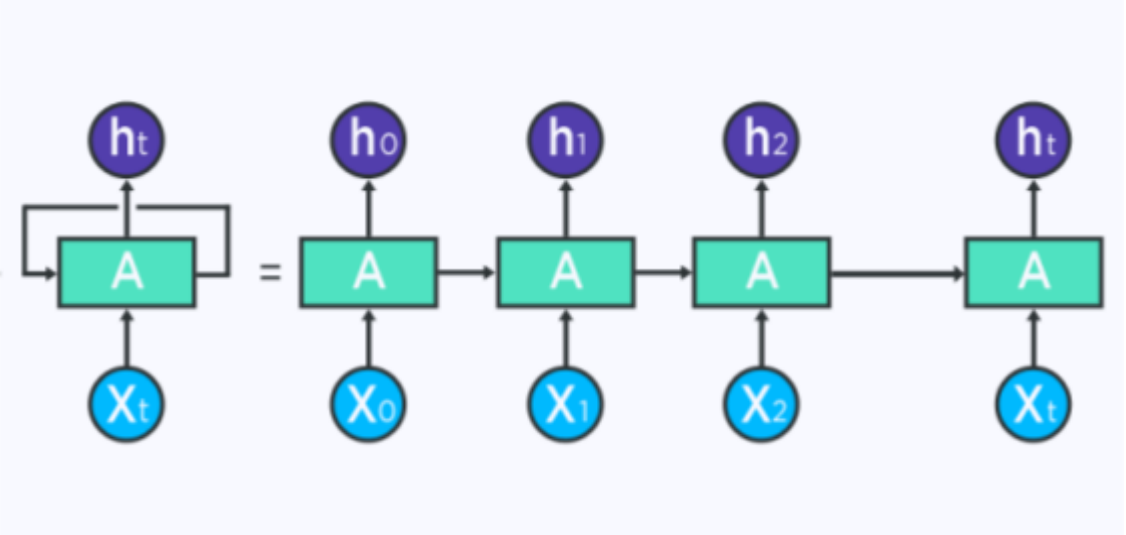

Рис. 2.3. Розвернута RNN

Основні проблеми стандартних RNN

Рекурентні мережі мають дві перешкоди з якими вони стикаються:

● Зростаючі градієнти – ситуація, в якій алгоритм з маловідомими причинами призначає великі значення для вагів, тим самим значно підвищуючи вплив вхідних даніх. Цю ситуацію можна виправити, застосувавши процедури усікання або стиснення градієнтів.

● Зникаючі градієнти – ситуація, в який значення градієнту занадто малі та модель припиняє навчання, або займає невиправдану кількість часу. Ця проблема є складнішою, ніж зростаючі градієнти, що призвело до винаходження LSTM нейронної мережі[9].

2.2 Нейронні мережі LSTM

Довга короткочасна пам’ять, LSTM (Long Short-Term Memory) – це розширення рекурентних нейронних мереж, де відбувається розширення пам’яті. Таким чином, вона краще підходить для вивчення важливих залежностей якій мають більшу відстань один від одного. З часом, було створено спрощену версії LSTM – GRU, яка має менше параметрів та вважалася деякий час більш продуктивною. Проте останні дослідження

показали, що LSTM більш ефективнішим.

Для порівняння схематичне зображення класичної RNN мережі:

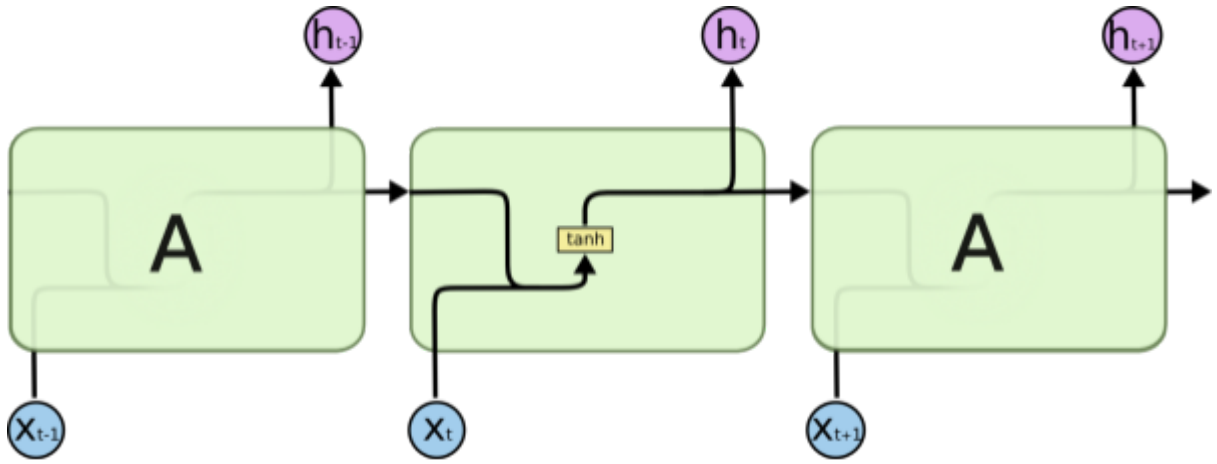

Рис. 2.4 Схематичне представлення класичної RNN

Вихід класичної RNN є рівняння:

$$h_t = tanh(W_{xh}^T x_i + W_{hh}^T h_{t-1} + b_h),$$

де tanh – тангенційна відцентрована функція активації, що дає значення в межах [-1;1], x_i – вектор вхідних даних, W_{xh}^T – вектор вагів для вхідних даних, h_{t-1} – вектор вихідних даних попередніх шарів, W_{hh}^T – вектор вагів для вектору вихідних даних попередніх шарів, b_h – попереднє зміщення.

LSTM мережі мають ланцюго-подібну структуру, але внутрішня структура має інший вигляд, маючи 4 нейронних шари, які взаємодіють між собою. Наступні позначення використані на схемі LSTM мережі:

Рис. 2.5. Використані позначення

LSTM нейронна мережа схематично зображується наступно:

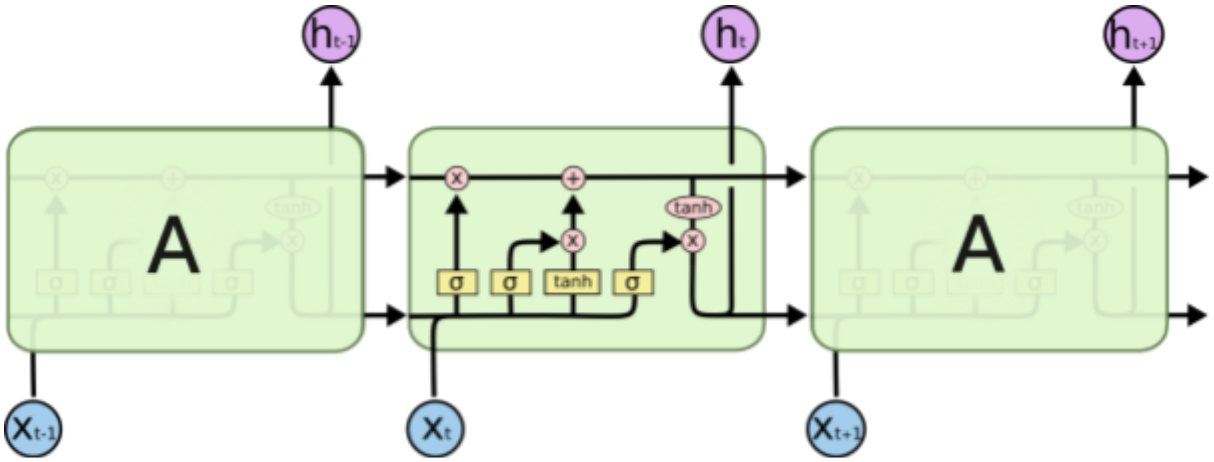

Рис. 2.6. Схематичне представлення LSTM

Ключовим компонентом LSTM є стан елементу, що є горизонтальна лінія, що проходить через весь нейрон на рисунок 2.7. Стан клітини є конвеєром всередині елемента з точками взаємодії, часто інформація проходить через елемент не зміненою.

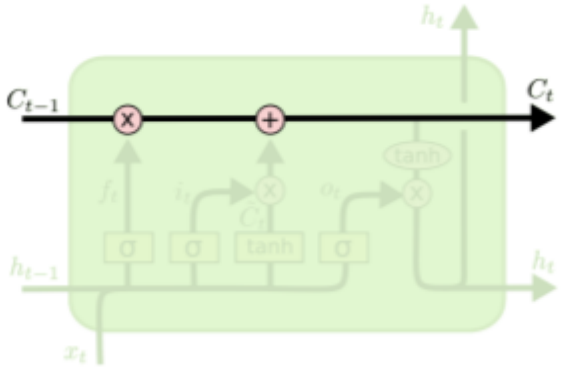

Рис. 2.7. Стан елементу

Тому LSTM має змогу добавляти або видаляти інформацію зі стану елемента шляхом відрегульованих структур, названі **затворами** (англ. gates).

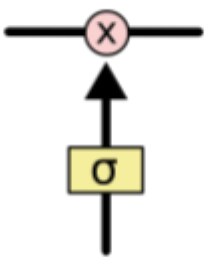

Рис. 2.8. Вихід (англ. gate)

Затвори називають динамічною пропускною структурою інформації, які складаються з нейронного шару з сигмоїдною активаційною функцією та операціями поточкового перемноження. Сигмоїдний шар дає на виході числа від 0 до 1, зазначаючи скільки кожному компоненту дозволено пройти: 0 – нічого не пропускати, 1 – пропусти все. Такий виходів існує три.

Покроковий розгляд LSTM структури

Перша компонента в LSTM вирішує яку інформацію потрібно видалити з стану елементу. Таке рішення приймає сигмоїдний шар названий **затвор забуття**. Враховуючи вектори h_{t-1} та x_i та обраховує значення в межах 0 та 1 для кожного значення вектору c_{t-1}.

Нехай дано часовий ряд поведінки користувачів на сервісі, припустимо, що за специфікою задачі, поведінка користувачів в будні дні відрізняється від поведінки у вихідні дні. В такому випадку мережі потрібно забути патерн поведінки будніх днів при передбаченні вихідних.

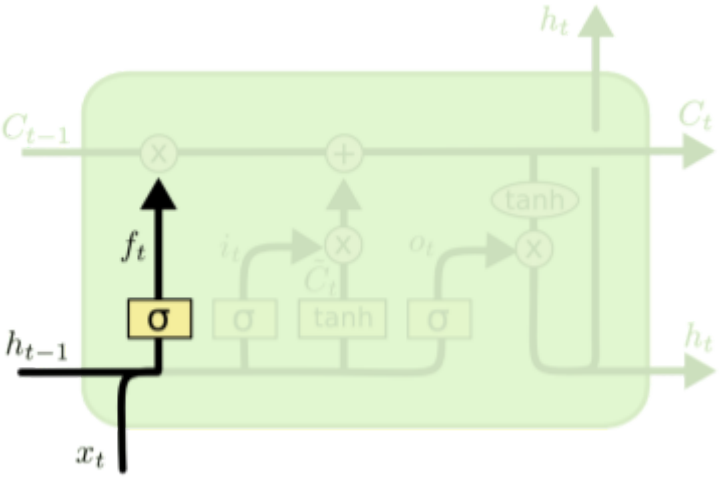

Рис. 2.9. Вихід забуття.

Вихід забуття є **нейроном**, який також знаходить вектор вагWдля відповідних x_t та h_t:

$$f_t = \sigma(W_{xf}^T x_t + W_{hf}^T h_{t-1} + b_f)$$

Другий компонент вирішує яку нову інформацію ми зберігатимемо в стані елементу. Перша складова є сигмоїдним шаром названий “вхідний затвор”, який вирішує які значення будуть оновлені. Друга складова є тангенційно активований шар, який створює вектор нових значень $\overline{C}_t$, враховуючи x_t та x_t та h_t,що будуть добавлятися до існуючого стану. Комбінація цих двох компонентів використовується, щоб оновити загальний стан елементу.

Даний об’єднаний компонент можна використати для пояснення процесу оновлення стану елементу, коли замінюємо патерн будніх днів на вихідні дні.

$$\overline{C}_t = tanh\ (W_{xc}^T x_t + W_{hc}^T h_{t-1} + b_c)$$

$$i_t = \sigma(W_{xi}^T x_t + W_{hi}^T h_{t-1} + b_i)$$

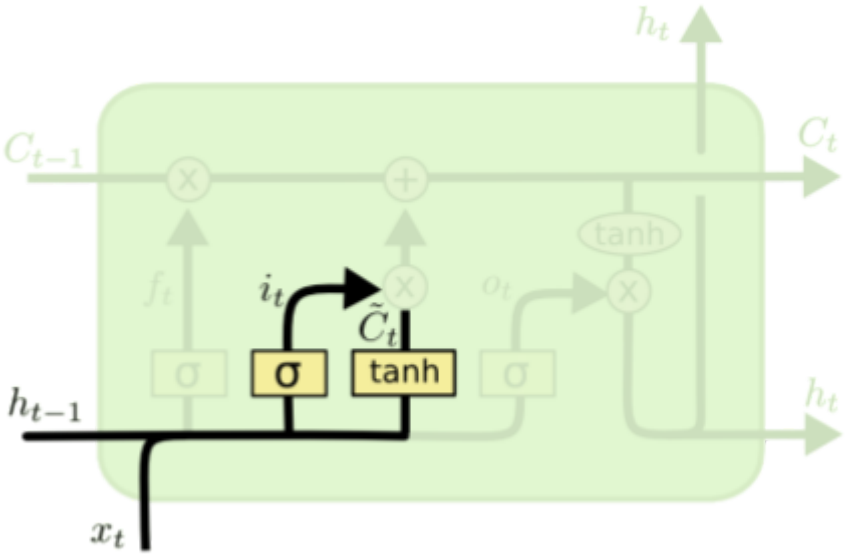

Рис. 2.9. Вхідний затвор оновлення

Щоб оновити попередній стан елементу C_{t-1}на новий C_t,в попередньому кроці було вирішено, що оновлювати. В цій процедурі буде

виконуватися дії для оновлення всього стану. Відбувається множення попереднього стану на f_t,забуваючи речі, які було вирішено забути в нейроні, де обраховується f_t. Сумуємо це значення до $i_t \cdot \overline{C}_t$та отримуємо нові значення, які змасштабовані відносно вирішеного оновлення на попередніх кроках.

$$C_t = f_t \odot C_{t-1} + i_t \odot tanh(W_{xc}^T x_t + W_{hc}^T h_{t-1} + b_c)$$

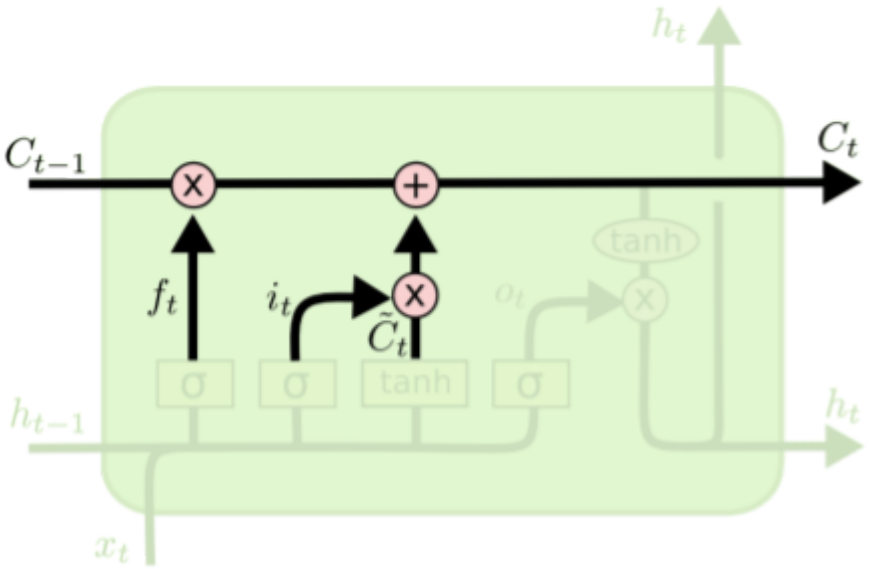

Рис. 2.10. Оновлення стану елемента

Останнім затвором вирішується які дані будуть передані на вихід, що є **вихідним затвором**. Цей вихід буде обчислюватися на базі стану клітини, але у відфільтрованому вигляді. Спершу запускається сигмоїдний нейронний шар, який вирішує які частини стану клітини потрібну подати на вихід. Далі стан клітини пропускається через **tanh** (щоб отримати значення між -1 та 1) та перемножається з вектором значень отриманим від вихідного затвору, таким чином пропустивши значення, які було вирішено пропустити в нейроні.

$$o_t = \sigma(W_{xo}^T x_t + W_{ho}^T h_{t-1} + b_o)$$

$$h_t = o_t \odot tanh(c_t)$$

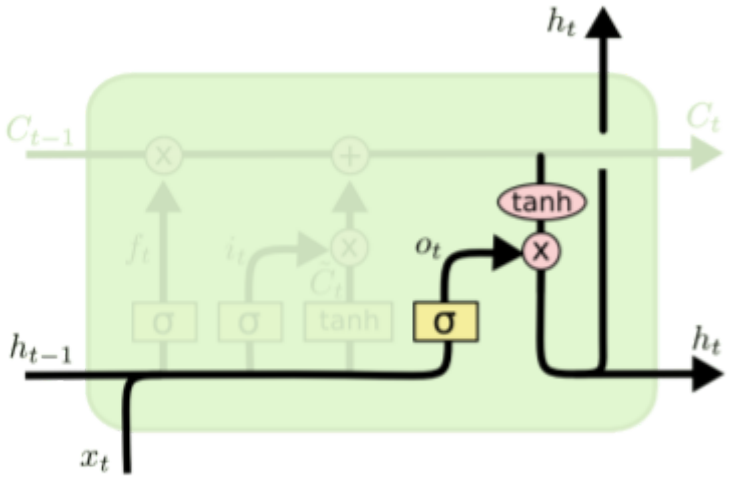

Рис. 2.11. Вихідні значення

На прикладі, нехай наступний елемент після цього є часові передбачення на початок будніх днів, тобто понеділка. Такий елемент має подати на вихід інформацію про закінчення вихідних, що означатиме підвищений шанс активності користувача.

Висновок

Було розглянуто принцип роботи рекурентних нейронних мереж на яких базується LSTM моделі. Це перша нейронна мережа, яка пам’ятає свої вхідні дані у внутрішній пам’яті, що відіграє ключову та невід’ємну роль у вирішенні проблем, що пов’язані з послідовними даними. Тому RNN віддається перевагу як алгоритму для послідовних даних як часових рядів, розпізнавання мови, тексту, фінансові дані, аудіо- та відеоряди, погода і в багатьох інших сферах. Було розглянуто принцип роботи RNN та прийдено проаналізовано її дві основні проблеми: **зростаючі** та **зникаючі** градієнти[10].

Розглянуто LSTM мережу, що дала значний прорив в часових передбачень. Вона краще підходить для вивчення важливих залежностей якій мають більшу відстань один від одного. Розглянуто поглиблено компоненти LSTM, її ключові затвори які допомагають контролювати попередні дані для наступних передбачень.

РОЗДІЛ 3

ВИБІР ТА ОБРОБКА НАБОРУ ДАНИХ ДЛЯ НАВЧАННЯ ТА ТЕСТУВАННЯ МОДЕЛІ

Часто дата сети отримуються в сирому та необробленому вигляді, збираючись для застосування в певному контексті поставленої задачі. Випадків, де не відбувається початкова обробка даних надзвичайно рідко трапляються, або поставлена задача є тривіальною, тому обробка даних набуло статусу правила при будуванні нейронної мережі. Три основні правила потрібно мати на увазі при побудові:

● Машинне навчання вимагає числові дані. Дані в датасеті часто приймають різних вигляд: цілі числа, числа з плаваючою точкою, рейтинг, відсотки, імена, категорії, позначки, булеві змінні та інші. Дані в таких алгоритмах працюють тільки з числами, оскільки дані представляються у вигляді векторів та матриць. Алгоритми машинного навчання можуть автоматично перетворювати дані, проте правила таких перетворень потрібно шукати в документації алгоритму.

● Кожен алгоритм машинного навчання вимагає різний тип даних та варто зробити дослідження, який алгоритм найкраще підходить під отримані дані та задачу. Наприклад, деякі алгоритми вимагають вхідні змінні, що відповідають Гаусівському розподілу. Також деякі можуть справлятися гірше, якщо присутні недоречні та лишні дані для цільових даних, які будуть виконувати роль шумів. Ефективність алгоритму може бути під негативним впливом від високо корельованих змінних, тому такі змінні мають бути ідентифіковані та видалені.

● Продуктивність алгоритму залежить від даних. Набір даних може бути “слабким представником” поставленої задачі і на це може бути дві причини. Перша причина – складні нелінійні співвідношення були стиснені

в необроблені дані, які можна отримати використовуючи техніки для підготовки даних. Друга – самі дані не є ідеальними, починаючи з помірно випадковими відхиленнями в спостереженнях, рахуючи їх до статистичного шуму, та до помилок, що призводять до значень, що виходять за припустимі межі, викликаючи конфлікти даних[11].

3.1 Використані дані в роботі

В роботі використано дані за 5 місяців (жовтень 2019 – лютий 2020) з eCommerce косметичного магазину. В необробленому вигляді дані розбиті помісячно та в сумі займають 2.27 ГБ місця. Дані містять ряд полів, які описують дії користувачів на сервісі такі як: перегляд товарів, покупка, добавлення в корзину та видалення з неї. Ряд даних в даному наборі є зайвими і вимагає очистки. Наступні поля будуть використані для алгоритму: **event_time** та **user_id**.

Також події можуть повторюватися декілька в даному наборі даних, тому буде зроблено припущення, що користувач виконував дії в півгодинному вікні. Оскільки занадто близькі значення можуть впливати на мережу як шуми, тому варто прийняти ключові точки на проміжку часу.

Дані в датасеті не є поділеними чи згрупованими по користувачам, що суперечить поняттю числовим рядам. Необхідно провести процедуру групування 30-и хвилинних вікон для кожного користувача. Також не всі користувачі виконували дії на сервісі регулярно, що не дасть змоги мережі вивчити їхню поведінку, тому потрібно знайти оптимальну кількість дозволених часових вікон, які пропускатиме користувачів для навчання експериментальним шляхом.

event_time	event_type	product_id	category_id	category_code	brand	price	user_id	user_session
2019-12-01 00:00:00 UTC	remove_from_cart	5712790	1.48758000526846E+018		f.o.x	6.27	576802932	51d85cb0-897f-48d2-918b-ad63965c12dc
2019-12-01 00:00:00 UTC	view	5764655	1.48758000541106E+018		cnd	29.05	412120092	8adff31e-2051-4894-9758-224bfa8aec18
2019-12-01 00:00:02 UTC	cart	4958	1.48758000947115E+018		runail	1.19	494077766	c99a50e8-2fac-4c4d-89ec-41c05f114554
2019-12-01 00:00:05 UTC	view	5848413	1.48758000767599E+018		freedecor	0.79	348405118	722ffea5-73c0-4924-8e8f-371ff8031af4

Рис. 3.1. Початковий вигляд даних.

3.2 Обробка даних з Python

Імпортуємо бібліотеку pandas для роботи з csv файлами та datetime об’єкт для його наступного використання при написанні лямбда функції, що буде описує як форматувати текстові дані в об’єкт datetime.

```
import pandas as pd
from datetime import datetime

dateparse = lambda x: datetime.strptime(x, '%Y-%m-%d %H:%M:%S
UTC')
```

Виконуємо зчитування даних з файлів помісячно в DataFrame, беручи тільки колонки *event_time* та *user_id*. Виконуємо перетворення рядку дати в об’єкт за допомогою лямбда функції *dateparse*. Та встановлюємо параметр index_col, щоб пропустити створення колонки як індексів DataFrame елементів. Виконуємо цю процедуру для кожного файлу окремо, щоб уникнути помилки завантаження, оскільки розміри файлів суттєві.

```
data1 = pd.read_csv('/content/drive/MyDrive/2019-Oct.csv',
   usecols=['event_time', 'user_id'],
   parse_dates=['event_time'],
   date_parser=dateparse,
   index_col=None)

data2 = pd.read_csv('/content/drive/MyDrive/2019-Nov.csv'...
data3 = pd.read_csv('/content/drive/MyDrive/2019-Dec.csv'...
data4 = pd.read_csv('/content/drive/MyDrive/2020-Jan.csv'...
data5 = pd.read_csv('/content/drive/MyDrive/2020-Feb.csv'...
```

Використовуючи метод pandas value_counts(), впевнюємося що дані було завантажено, а також бачимо кількість подій для кожного користувача в одній з п’яти змінних.

```
data4['user_id'].value_counts()
```

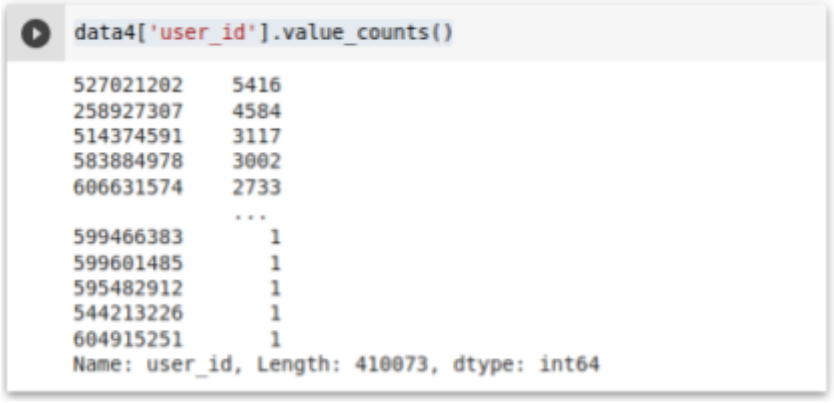

Рис. 2.2. Результат перевірки даних

Файли зберігаються в сервісі Google Disk, оскільки Google Colaboratory може швидко підключатися до нього та завантажувати звідти файли, пікдлючивши модуль:

```
from google.colab import files
```

Розглянемо функцію трансформації даних. В ній використано почастинне розбиття даних та запис їх у файл, оскільки ця процедура займає 25-30 хвилин для кожного файлу, важливо мати можливість моніторити працездатність процесу.

Ініціалізуємо початок функції, що приймає DataFrame, як аргумент. Отримуємо кількість рядків в цьому фреймі даних та ставимо крок після якого буде відбуватися запис згрупованих даних у файл та початковий показник.

```
def transform_data (df):
    total_rows = df.shape[0]
    step = 50000
    pointer = 0
```

Створюємо цикл, який на перший погляд буде виконуватися нескінченно, проте для цього випадку ми використовуємо змінну pointer, яка постійно інкрементується та при рівності зі змінною кількості рядків у файлі total_rows закінчує роботу циклу.

Виконуємо важливу перевірку, щоб не втратити частину даних при

кінцевих записах. Якщо при наступній ітерації показник *pointer* плюс наступний крок *step* більше кількості рядків, потрібно знайти крок, коли це значення ще не є більшим за *total_rows* шляхом:

```
 if (pointer + step >= total_rows):
           step = total_rows - pointer
```

Далі виконуємо ключову операцію групування за допомогою методу *groupby() pandas* бібліотеки, що дає можливість розбити дії кожного користувача на ключові 30-и хвилинні вікна та відкинувши поєднані дублікати з *user_id*. Виконуємо перейменування колонок та скидаємо індекси внаслідок групування, виконуємо запис у файл на Google Disk.
Отримаємо зміст циклу:

```
while (True):
       if (pointer + step >= total_rows):
           step = total_rows - pointer

       grouped =
data5[pointer:pointer+step].groupby(['user_id',
pd.Grouper(key='event_time', freq='30Min')]).agg(lambda
subgroup: subgroup.drop_duplicates(subset='user_id'))
       grouped = grouped.rename(columns={'event_time': 'time',
'user_id': 'user'})
       grouped.reset_index(inplace=True)
       grouped.to_csv('/content/drive/MyDrive/Grouped
data/data-Feb.csv', mode='a', columns=['event_time',
'user_id'], header=None, index=None)
       print('+step', pointer+step)

       pointer += step

       if (pointer == total_rows):
           return
```

Дані було погруповано в 30-и хвилинні проміжки з кроком 50000 тисяч рядків, тому можлива ситуація повторення цих 30-и хвилинних проміжків, оскільки близькі події користувача можуть лежати на межах кроку.

Зробимо фінальне очищення від дублікатів:

```
cleaned_data1 = pd.read_csv('/content/drive/MyDrive/Grouped
data/data-Feb.csv')
cleaned_data1.drop_duplicates()
cleaned_data1.to_csv('/content/drive/MyDrive/Cleaned
data/cleaned_data_Feb.csv', mode='w', header=None, index=None)
```

Очищені дані матимуть вигляд:

2019-12-01 09:30:00	48182054
2019-12-01 10:00:00	48182054
2019-12-01 11:00:00	59738436
2019-12-01 05:00:00	69948697
2019-12-01 09:30:00	72062431
2019-12-01 10:00:00	72062431
2019-12-01 06:00:00	72655920
2019-12-01 06:30:00	72655920
2019-12-01 07:30:00	72655920
2019-12-01 08:00:00	72655920
2019-12-01 09:00:00	72655920
2019-12-01 07:30:00	73922615
2019-12-01 08:00:00	73922615
2019-12-01 01:00:00	80310232
2019-12-01 11:00:00	80841298

Рис. 2.3. Результат очищених даних

Зараз дані розкидані на 30-и хвилинні вікна без групування по користувачам. Потрібно отримати дані для кожного користувача. Імпортуємо системні модулі *os* та *glob*, щоб отримати доступ до файлів операційної системи машини, яка є хостом для поточного середовища. Для конвертування дат зробимо імпорт *datetime*, для роботи з дата фреймами *pd*, для високопродуктивних операцій з числами використаємо *np*, *pickle* для запису об’єкту у файл.

```
import os
import glob
from datetime import datetime
import pandas as pd
import numpy as np
import pickle
```

Зробимо об’єднання шляхів до файлів та завантажимо файли однією процедурою, оскільки тепер дані в межах 100 Мб. Замінимо дати на Unix числові вирази, щоб дати нейронній мережі проводити математичні операції.

```
path = r'/content/drive/MyDrive/Cleaned data/'
all_files = glob.glob(os.path.join(path, "*.csv"))

df_from_each_file = (pd.read_csv(f, parse_dates=[0],
names=['event_time', 'user_id'],) for f in all_files)
c_df   = pd.concat(df_from_each_file)

c_df['event_time'] = c_df.event_time.values.astype(np.int64)
// 10 ** 9
```

На початковому етапі візьмемо поріг допуску користувачів до навчання 500, далі експериментальним шляхом можна буде визначити оптимальне значення. Відфільтруємо користувачів за допомогою функцій *value_counts()*, яка вказує кількість дат для кожного користувача та за допомогою *isin()* залишимо тільки ті записи, які належать до відфільтрованих користувачів. Отримаємо унікальні значення ідентифікаторів в змінну *USER_IDS*.

```
count_threshold = 500

counts = c_df[ 'user_id' ].value_counts()
filtered_df = c_df[c_df['user_id'].isin(counts[count_threshold
> 500].index)]
USER_IDS = np.unique(filtered_df.index.values)

print('\nВсього користувачів:', USER_IDS.size)
```

З даним пропускним значення отримуємо 18 користувачів здатних до вивчення їхньої поведінки. Створюємо об’єкт в якому будуть числові ряди для кожного користувача окремо. Записуємо об’єкт в pickle файл, який дає швидко імпортувати та експортувати об’єкти типу dictionary.

```
TIMESERIES_PER_USER = {}

for user_id in USER_IDS:
   timeseries = filtered_df.loc[ user_id , 'event_time' ]
   TIMESERIES_PER_USER[int(user_id)] = timeseries.values

with open('/content/drive/MyDrive/timeseries.pickle', 'wb') as
handle:
   pickle.dump(TIMESERIES_PER_USER, handle,
protocol=pickle.HIGHEST_PROTOCOL)
```

Перевіряємо правильність виконання, пробуючи завантажити об’єкт з файлу:

```
infile = open('/content/drive/MyDrive/timeseries.pickle','rb')
timeseries = pickle.load(infile)

timeseries
```

```
{72655920: array([1569916800, 1569949200, 1570003200, 1570005000, 1570010400,
        1570014000, 1570023000, 1570035600, 1570084200, 1570091400,
        1570093200, 1570095000, 1570096800, 1570102200, 1570116600,
        1570174200, 1570190400, 1570203000, 1570264200, 1570278600,
        1570280400, 1570282200, 1570354200, 1570356000, 1570365000,
        1570428000, 1570433400, 1570446000, 1570532400, 1570550400,
        1570613400, 1570615200, 1570617000, 1570633200, 1570635000,
        1570701600, 1570786200, 1570798800, 1570860000, 1570876200,
        1570957200, 1570959000, 1570977000, 1571131800, 1571133600,
        1571149800, 1571212800, 1571223600, 1572420600, 1572433200,
        1572436800, 1572510600, 1572514200, 1572589800, 1572591600,
        1572604200, 1572607800, 1572609600, 1572613200, 1572615000,
```

Рис. 2.4. Дані отримані з pickle файлу

Отримані дані представлені у вигляді об’єкту з ключ-значеннями у вигляді user_id: array[dates]. Такі дані готові бути переданими в нейронну модель для навчання, перетворивши їх у числові значення, розбивши на кожного користувача, з видаленими зайвими даними та розділеними на 30-и хвилинні вікна.

Висновок

На даному етапі було проведено процедуру вибору та обробки даних для подальшого їх передачі для навчання та тестування даних. Було обрано набір даних з eCommerce бази даних, в якій знаходиться активність користувачів на сервісі. Було розділено набори активностей на 30-и хвилинні вікна, що значно спростить навчання мережі LSTM та її більш стійкі результати передбачення. Також було проведено процедуру групування даних по користувачам, оскільки навчання відбувається для кожного користувача, оскільки поставлена задача вказує на персоналізовані сповіщення користувачів. Обов’язково було очищено дублікати 30-и хвилинних вікон, оскільки вони можуть давати неправдиву дані для мережі, використовуючи модулі з бібліотек Numpy, Pandas мови програмування Python, що пропонують ряд утиліт та значно оптимізовані операції зі значеннями, оскільки змінні Python представлені об’єктами та є об’ємними за допомогою своїх методів.

РОЗДІЛ 4

ПОБУДОВА МОДЕЛІ ПЕРЕДБАЧЕННЯ ЧАСОВИХ РЯДІВ LSTM КЛАСУ ДЛЯ ЇЇ ПОДАЛЬШОГО ЗАСТОСУВАННЯ У ВІДПРАВЦІ СПОВІЩЕНЬ

Для побудови нейронної мережі LSTM необхідно розділити датасет на навчання мережі та її тестування. Що дасть змогу виконати процедуру перевірки ефективності роботи. Для підняття ефективності та швидкості навчання мережі застосовують нормалізацію для даних, що розташовує дані на проміжку [0,1] за допомогою масштабування.

Виконуємо імпорт необхідних модулі для роботи, раніше не застосовані модулі такі як Sequential, LSTM, Dense, tf є модулями для побудови нейронної мережі. Модуль Sequential використаний для побудови декількох послідовних LSTM шарів та використано Dense для вихідного значення, оскільки при навчанні, зворотне поширення помилки починається з вихідного шару. Імпортуємо MinMaxScaler для масштабування даних для тренування та тестування.

```
from numpy import array
import pickle
import numpy as np
import pandas as pd
from keras.models import Sequential
from keras.layers import LSTM
from keras.layers import Dense
import tensorflow as tf
from pandas import Series
from sklearn.preprocessing import MinMaxScaler
```

4.1 Розділення даних для навчання

Підготуємо функцію для розділення даних яка розбиватиме дані для тренування та тестування. Функція приймає три параметри числовий ряд series, кількість попередніх кроків *steps_to_learn*, щоб враховувати при навчанні, тобто скільки минулих кроків потрібно зберігати в пам’яті, щоб виконати передбачення для наступних *steps_to_predict*. Створюємо X, Y списки, які будуть містити числові підряди з довжиною *steps_to_learn* та *steps_to_predict* відповідно. Вхідний параметр series

сортується по зростанню, щоб отримати послідовність. Щоб отримати яку кількість вибірок можливо отримати з даною числовою послідовністю використаємо вираз:

```
samples_count = len(series) - steps_to_learn - steps_to_predict + 1
```

За допомогою змінної *end_index* буде відбуватися контроль межі, який буде слугувати розділом коли значення класифікувати в список змінних для вивчення та передбачення.

```
def series_splitter(series, steps_to_learn, steps_to_predict):
   X, Y = list(), list()
   series.sort()
   samples_count = len(series) - steps_to_learn -
steps_to_predict + 1
   for i in range(samples_count):
       end_index = i + steps_to_learn
       end_predict_index = end_index + steps_to_predict
       series_x = series[i:end_index]
       series_y = series[end_index:end_predict_index]
       X.append(series_x)
       Y.append(series_y)
   return array(X), array(Y)
```

Завантажуємо раніше записаний файл з об’єктом, де ключ-значення ідентифікатор користувача до списку його числових послідовностей з pickle.

```
user_timeseries_file =
open('/content/drive/MyDrive/timeseries.pickle','rb')
user_timeseries = pickle.load(user_timeseries_file)
```

traintest_border – змінна, яка є експериментальним гіперпараметром для розділення межі між даними для навчання та тестування. Виберемо одного користувача з декількох та розділимо його числовий ряд.

```
traintest_border = 500
dataset = array(user_timeseries[72655920])
XY_train = dataset[:traintest_border]
XY_test = dataset[traintest_border:]
```

4.2 Нормалізація даних

Виконаємо невід'ємну процедуру масштабування даних, що на виході дасть відносні значення в межах [0, 1]. Таке перетворення необхідне для зміни числових значень в датасеті до спільного масштабу без спотворення різниці між цими значеннями. При спотворенні різниці відбувається втрата цінної інформації для моделі. Нормалізація необхідна для ряду алгоритмів для правильної їхньої роботи, оскільки великі різниці в масштабах можуть створювати конфлікти та некоректність, наприклад, якщо алгоритм використовує сигмоїдну функцію.

Для коректної нормалізації необхідно перетворити масив даних в форму `values.reshape((len(values), 1))`, що означає *len(values)* рядків по 1 елементу. Викличемо функцію *fit()* для отримання мінімуму та максимуму для майбутнього масштабування *transform()*, функція яка саме виконує масштабування всіх значень. Виконаємо цю процедуру для даних навчання та тестування та отримаємо масштабовані масиви даних у формі *(1, len(values))*.

```
XY_train_series = Series(XY_train)
# print(XY_train_series)
# prepare data for normalization
values = XY_train_series.values
values = values.reshape((len(values), 1))
# train the normalization
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(values)
# print('Min: %f, Max: %f' % (scaler.data_min_,
scaler.data_max_))
# normalize the dataset and print
XY_train = scaler.transform(values).flatten()

# normalize test data
XY_test_series = Series(XY_test)
# print(XY_train_series)
# prepare data for normalization
values = XY_test_series.values
values = values.reshape((len(values), 1))
# train the normalization
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
scaler = scaler.fit(values)
# print('Min: %f, Max: %f' % (scaler.data_min_,
scaler.data_max_))
# normalize the dataset and print
XY_test = scaler.transform(values).flatten()
```

Виберемо гіперпараметри *n_steps_in*, *n_steps_out* експериментальним шляхом, які будуть коригуватися для отримання найкращих результатів. *n_steps_in* – кількість часових значень LSTM триматиме в пам'яті для врахування передбачення *n_steps_out* часових значень.

Виконуємо процедуру розділення даних навчання та тренування на вхідні дані *X_train*/*X_test* та правдиві вихідні дані *Y_train*/*Y_test*. Присвоємо значення змінній *n_features = 1*, оскільки в даному випадку використано однозмінні дані. Якщо, наприклад, враховувалося б чи користувач виконав покупку, оскільки покупка може прямо вплинути на його майбутню активність, та коли він її виконав, можна використати мультизмінні вхідні дані (*n_features = 2*).

Виконаємо перетворення форми даних навчання та тестування у форму, яку приймає LSTM модель. Ця форма має вигляд $S \times N \times D$, де S – кількість зразків, що дає виході функція *series_splitter()*, N – кількість вхідних змінних, або кроки для запам'ятовування, D – 1 у випадку однозмінних вхідних значень та записується наступно:

```
    X_train.reshape((X_train.shape[0], X_train.shape[1],
n_features))
```

Повна процедура розділення та перетворення наступна:

```
n_steps_in, n_steps_out = 12, 4
X_train, Y_train = series_splitter(XY_train, n_steps_in,
n_steps_out)
X_test, Y_test = series_splitter(XY_test, n_steps_in,
n_steps_out)
n_features = 1
X_train = X_train.reshape((X_train.shape[0], X_train.shape[1],
n_features))
X_test = X_test.reshape((X_test.shape[0], X_test.shape[1],
```

```
n_features))
print(X_train.shape, X_test.shape)
```

4.3 Побудова моделі

Використаємо модуль Sequential, який дасть змогу послідовно добавити елементи в модель нейронної мережі. Добавимо LSTM модуль послідовно два рази з формою вхідних даних (n_steps_in, n_features), що в даному випадку рівно (12, 1). Також вкажемо return_sequences = True, що вихідні дані LSTM потрапляли на вхід іншої LSTM компоненти. В обох випадках використаємо активаційну функцію relu та виберемо експериментальний гіперпараметр units, що йде перший параметром в моделі рівною 50. Добавимо Dense шар, який на виході дасть 4 значення без активаційної функції. При компіляції моделі використаємо часто використаний оптимізатор adam, який є покращеним алгоритмом відносно стохастичного градієнтного спуску, який є ключовою складовою знаходження вагів вхідних даних, а також використаємо функцію втрати MSE (середню квадратичну похибку).

```
model = Sequential()
model.add(LSTM(50, activation='relu',
return_sequences=True, input_shape=(n_steps_in,
n_features)))
model.add(LSTM(50, activation='relu'))
model.add(Dense(n_steps_out))
model.compile(optimizer='adam', loss='mse')
```

Викликаючи команду *model.summary()* отримаємо структуру моделі, кожен її шар, їх форму вихідних даних та кількість параметрів кожен шар повинен вивчити. Через те, що перший шар повертає послідовності на вхід другого шару.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 12, 50)            10400
_________________________________________________________________
lstm_1 (LSTM)                (None, 50)                20200
_________________________________________________________________
dense (Dense)                (None, 4)                 204
=================================================================
Total params: 30,804
Trainable params: 30,804
Non-trainable params: 0
```

Рис. 4.1. Структура побудованої моделі

Викликаємо процедуру *model.fit()*, що ініціює процедуру навчання моделі з масштабованими даними навчання *X_train* та *Y_train*, тестування *X_test* та *Y_test*. З кількістю гіперпараметром *epochs* (епохи) рівним 500. Гіперпараметр *epochs* необхідно обрати як найвищим до тих пір поки результативне значення *validation_loss* не почне зростати або довгий час залишатиметься на місці. Небезпека високого значення epochs впливає на перевивчення даних, коли модель починає навчатися шумам також. Використаємо гіперпараметр *batch_size=32*, що вказує на розмір партії зразків протягом однієї епохи *(epoch)*.

```
fit = model.fit(X_train, Y_train, epochs=1000, batch_size=32,
validation_data=(X_test, Y_test))
```

4.4 Отримання результатів та підсумок

За допомогою наступного коду отримаємо історію навчання моделі, викликавши метод fit.history() з відповідним обраними параметрами loss та val_loss. Відразу передамо ці історії вхідним аргументом для функції plt.plot() для побудови графіків.

```
# Plot loss per iteration
import matplotlib.pyplot as plt
plt.plot(fit.history['loss'][2:], label='loss')
plt.plot(fit.history['val_loss'][2:], label='val_loss')
plt.show()
```

Після виконання коду отримаємо наступний графік:

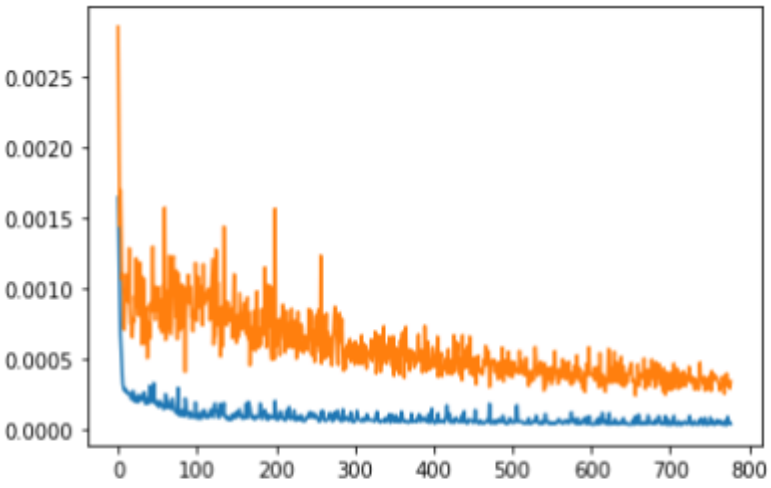

Рис. 4.2. Ключові показники навчання моделі

Оскільки дані при навчанні поділяються на два призначення – навчання та тестування. Ключовим показником є втрати (loss) даних навчання та тестування. Відносно цих втрат відбувається зворотне поширення змін у вагах кожного нейрону. На першому раунді навчання беруться випадкові ваги та поступово коригуються на наступних. Різниця між випадковими початковими вагами та кінцевими вагами можна чітко побачити на рисунок 4.3. На рисунок 4.2 зображено втрати навчання та тестування відкинувши перших два значення, для отримання масштабованих значень та візуального представлення навчання моделі.

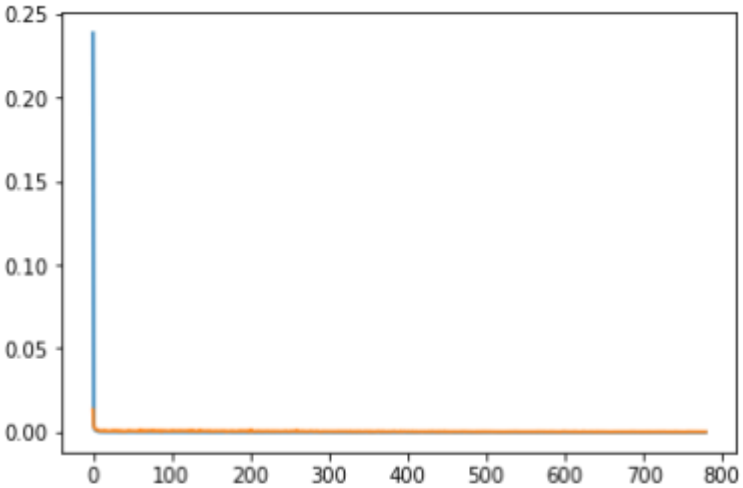

Рис. 4.3. Ключові показники з початковими випадковими вагами

За допомогою наступного коду виконаємо передбачення, використовуючи побудовану модель на базі LSTM. Для отримання однакового масштабу на графіку та однакового співставлення правдивих даних, які вже є зарані відомими, що є даними тестування, з відповідними передбаченнями моделі. З самого початку, призначимо значення змінній *validation_target* рівній кожному четвертому значенню валідаційних даних, уникнувши повторення значення, оскільки *Y_test* залежить від 12-и значень *X_test* та кожне значення *Y_test* є результатом зміщення на одиницю спільного датасету *XY_test*, яке ділиться за допомогою функції *series_splitter*.

Призначимо fourstep_shift = 0 як вхідне значення індексу. Створимо цикл який працюватиме до тип пір поки fourstep_shift менше довжини X_test. В циклі зробимо передбачення в побудованій моделі та зробимо додавання отриманих передбачень, що є 4 часових значення відмасштабованих відносно всього набору даних в межах [0, 1]. Додамо до fourstep_shift значення, щоб виконати наступних чотири передбачення на базі правдивих значень, а не на попередньо передбачених моделлю даних.

```
# Передбачення декількох шляхів
validation_target = Y_test[::4]
validation_predictions = np.array([])

# перше валідаційне значення
fourstep_shift = 0

while fourstep_shift < len(X_test):
   last_x = X_test[fourstep_shift]
   p = model.predict(last_x.reshape(1, n_steps_in,
1)).flatten()
   validation_predictions =
np.concatenate((validation_predictions, p))
   fourstep_shift += 4

plt.plot(validation_predictions, label='forecast prediction')
plt.plot(validation_target.flatten(), label='forecast target',
alpha=0.2)
plt.legend()
```

Виведемо графіки масивів передбачень та зарані відомих значень:

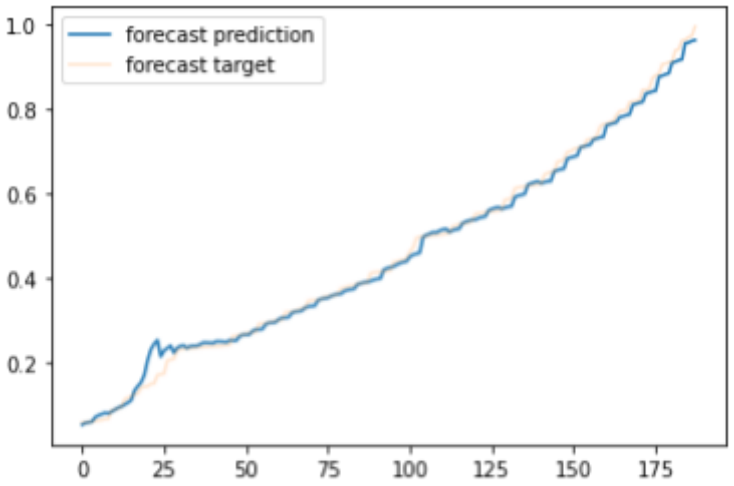

Рис. 4.4. Графік чотири-крокових передбачень моделі LSTM

На рисунок 4.4 показано графік передбачення чотири-крокових передбачень відносно зарані відомих тестувальних даних.

Підсумовуючи, було побудовано модель LSTM, яка здатна робити чотири-крокові передбачення майже насаджуючись поверх відомих правдивих значень *target forecast*. Для отримання кращих результатів пропонується виконати множину експериментальних змін в гіперпараметрах моделі для знаходження найефективнішої моделі для передбачення та швидкості навчання моделі. Для цього крім зміни гіперпараметрів добавляють Bidirectional LSTM, або Dropout елементи задля уникнення перенавчання моделлю даних. Дана модель може бути розширена до мультизмінності, тобто додавання інших параметрів або сутностей, які допомагають підвищенню точності передбачення, такі як чи користувач здійснив покупку, що може вказати на його збільшений час повертання на сервіс, тоді як йому можна було б “нагадати”.

ЗАГАЛЬНІ ВИСНОВКИ ПО РОБОТІ

1) Проведено аналіз існуючих методів передбачення часових рядів в статистичному аналізі та в машинному навчанні, а також порівняння методів статистичного аналізу з методами машинного навчання. Приведено слабкі сторони та припущення кожної з моделей, розглянувши базові принципи їх роботи.
2) Було досліджено принцип роботи рекурентних нейронних мереж та нейронних мереж в цілому. Можливі проблеми нейронних мереж та ключові компоненти, як нейрони, активаційні функції та їх різновиди. Розглянуто головний принцип роботи LSTM класу нейронних мереж на аналітичних виразах, так і схематичне візуальне їх представлення на рисунках.
3) Було обрано підходящий набір даних одного з сервісу eCommerce сфери для навчання моделі. За допомогою Python було зроблено групування даних у 30-ти хвилинні вікна без дублікатів та згруповано для кожного користувача. Також виконано процедуру нормалізації для пришвидшеного та одно масштабного навчання, що вимагає LSTM в даному випадку, оскільки числові значення досягають мільйони, які потім пропускаються через сигмоїдні активаційні функції.
4) Побудовано модель передбачення часових рядів за допомогою класу LSTM, яка здатна передбачати чотири кроки наперед, що і є ціллю роботи – виконувати багатокрокові передбачення активності користувача задля економії обчислювальних ресурсів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Bzdok, D., Altman, N. & Krzywinski, M. Statistics versus machine learning. *Nat Methods* **15,** 233–234 (2018). https://doi.org/10.1038/nmeth.4642
2. Eugine Kang. Hidden Markov Model. https://medium.com/@kangeugine/hidden-markov-model-7681c22f5b9
3. kbrose (https://stats.stackexchange.com/users/49646/kbrose), Hidden Markov Model vs Recurrent Neural Network. https://stats.stackexchange.com/q/366430
4. Kowshik Chilamkurthy, Understanding Point Processes. https://kowshikchilamkurthy.medium.com/understanding-point-processes-6e3d2f6c5480
5. Kowshik Chilamkurthy, Neural Hawkes Process. https://medium.com/point-processes/neural-hawkes-process-1a954136a078
6. Abhishek Rajbhoj, ARIMA simplified. https://towardsdatascience.com/arima-simplified-b63315f27cbc
7. Vandit Jain, Everything you need to know about “Activation Functions” in Deep learning models. https://towardsdatascience.com/everything-you-need-to-know-about-activation-functions-in-deep-learning-models-84ba9f82c253
8. Sumit Saha, A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53
9. Niklas Donges, A Guide to RNN: Understanding Recurrent Neural Networks and LSTM. https://builtin.com/data-science/recurrent-neural-networks-and-lstm
10. Christopher Olah, Understanding LSTM Networks https://colah.github.io/posts/2015-08-Understanding-LSTMs
11. Jason Brownlee, Why Data Preparation Is So Important in Machine Learning. https://machinelearningmastery.com/data-preparation-is-important