Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

Застосування методів машинного навчання для прогнозу часових рядів

Текстова частина до курсової роботи

за спеціальністю «Інженерія програмного забезпечення»- 121

Керівник курсової роботи

Кандидат технічних наук,
доцент
Олецький О.В.

_____

(підпис)

“____” _____ 2021 р.

Виконав студент ІПЗ-4:

Пархоменко Д.О.

“____” _____ 2021 р.

Київ-2021

Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ«КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри мультимедійній систем,

доцент, к.ф.-м.н.

_____О.П. Жежерун

“___” _____2021 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ
на курсову роботу

студенту Пархоменку Данилу Олександровичу

факультету Інформатики четвертого курсу

ТЕМА: Застосування методів машинного навчання для прогнозу часових рядів

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Анотація

Вступ

1. Алгоритм зворотного поширення похибки.
2. Архітектура нейронних мереж з довгою короткочасною пам’яттю.
3. Отримані результати з прогнозування часових рядів.
4. Порівняння з іншими способами прогнозування.

Висновки

Список літератури

Додатки (за необхідністю)

Дата видачі “___” _____ 2021 р. Керівник_____
(підпис)

Завдання отримав_____
(підпис)

ТЕМА: Застосування методів машинного навчання для прогнозу часових рядів
Календарний план виконання роботи:

No п/п	Назва етапу курсової роботи	Термін виконання етапу
1.	Отримання завдання на курсову роботу	15.10.2020
2.	Ознайомлення з методами вирішення задачі	20.12.2020
3.	Налаштування середовища у PyCharm та ініціалізація проекту	10.01.2021
4.	Перші спроби реалізації алгоритму зворотного поширення похибки	26.02.2021
5.	Застосування алгоритму для прогнозування часових рядів	14.03.2021
6.	Зміни програми на реалізацію на основі мережі з довгою короткочасною пам’яттю	25.03.2021
7.	Порівняння прогнозування з алгоритмом ARIMA	7.04.2021
8.	Написання текстової частини курсової роботи	
9.	Узгодження курсової роботи з науковим керівником	09.04.2021
10.	Написання презентації для захисту курсової роботи	11.04.2021
11.	Захист курсової роботи	16.04.2021

Студент Пархоменко Данило Олександрович

Керівник Олецький Олексій Віталійович

“_____”_____

Зміст

Анотація

У даній курсовій роботі описані загальні концепції штучних нейронних мереж. Продемонстрований спосіб застосування штучних мереж для вирішення регресивних задач різного роду. Також був проведений аналіз даних, що був обрані для подальшого прогнозування. Були зазначені обмеження та недоліки, які має алгоритм зворотного прогнозування помилок і наведені випадки, для яких потужності одного цього алгоритму бракує.

В кінці курсової додатково було порівняно результати з регресивним методом прогнозування, а саме з алгоритмом ARIMAX.

В роботі використовується такі засоби: Python 3.9, Tensorflow, Pandas, Statsmodel, Numpy, Matplotlib, Sklearn.

Ключові слова: Python, Часові ряди, Штучні нейронні мережі, Алгоритм зворотного поширення похибки, ARIMA, Tensorflow, Регресія, Мережі з довгою короткостроковою пам’яттю.

Вступ

Тема прогнозування часових рядів є дуже актуальною на наш час, часові ряди існують всюди будь то продажі певного магазину продовольчих товарів, врожайність пшениці на полях за останні десять років чи навіть коливання цін акцій на біржі. При застосуванні правильних інструментів прогнозування всі ці показники в теорії можна досить точно передбачувати, з похибкою меншою за десять відсотків або може навіть один відсоток. Завдяки такому прогнозуванню людина, яка не має аналітичних знань може досить точно визначати майбутні тенденції, що чекають на її бізнес чи організацію.

Звісно таке прогнозування не може передбачити різні інфляції, пов’язані з випадковими процесами, такі як природні катаклізми або пандемією, яка є дуже актуальна на даний момент. Гадаю, що передбачити такі інфляції буде не можливо, що найменш у цьому столітті.

З іншого боку, якщо інфляція буде сезонною, тобто іншими словами відбуватись з певною періодичністю, наприклад, п’ять, десять років, то це можливо спрогнозувати, якщо тренувати систему на проміжку, який більший або дорівнює проміжку між інфляціями.

Мета даної курсової роботи – з’ясувати, які є складності у прогнозуванні часових рядів, з якими може зіштовхнутись програміст, що намагається зробити систему для прогнозування часових рядів, також дослідити різні підходи до отримання потрібних результатів, я саме підхід з використанням штучних нейронних мереж та підхід з використанням лінійної регресії.

Результатом курсової роботи є система, що являє собою основу для прогнозування часових рядів. Для наочності на даний момент система може прогнозувати кількість пасажирів на рейсах однієї авіакомпанії за певний період часу, але в принципі, система може прогнозувати також інші числові значення, необхідно лише привести дані до потрібного системі вигляду.

Робота складається з чотирьох розділів.

В першому розділі описано особливості роботи алгоритму зворотного поширення похибки. Описані основні складові алгоритму та принцип його роботи, майже всі етапи роботи супроводжуються ілюстраціями, які допомагають детальніше зрозуміти концепцію алгоритму.

В другому розділі розглянуто штучну нейронну мережу зі структурою long short-term memory. Якраз завдяки якій прогнозування стало можливим. Ця нейронна мережа працює на основі алгоритму зворотного поширення похибки, але дозволяє уникнути недоліки, які має чистий алгоритм.

В третьому розділі описана сама система, що вийшла в результаті курсової роботи. Поетапно розглянуті всі кроки, які були необхідні для самого прогнозування. Також наведено статистичні дані про датасет, який використовувався у ролі реальних даних для прогнозування та про результати прогнозування, які дозволяють краще зрозуміти наскільки вдало працює система.

Четверту частину присвячено порівнянню прогнозування на основі штучних нейронних мереж з прогнозуванням на основі регресивного алгоритму ARIMA. Було порівняно точність прогнозування і складність реалізації обох підходів. В кінці розділу висловлена суб’єктивна думка автора з приводу того, який підхід все ж таки краще використовувати.

Задача курсової роботи виглядає наступним чином:

1. Взяти певні дані, що підпадають під характеристику часових рядів (продажи магазину, коливання цін акцій, кількість пасажирів на рейсах, кількість відвідувачів в аквапарк тощо).
2. Привести дані до правильного вигляду, тобто виділити тільки значення з потрібного проміжку, масштабувати значення до проміжку від нуля до одиниці)

3. Натренувати систему на тренувальних даних до моменту поки система не почне видавати досить точні результати
4. Запустити прогнозування вже на тестових даних.

Алгоритм зворотного поширення похибки.

Загальні відомості

Вперше алгоритм зворотного розповсюдження похибки був запроваджений в 1970-х роках, до відомого допису 1986 року Девіда Румельхарта, Джеффрі Хінтона та Рональда Вільямса важливість алгоритму не була повністю оцінена. У роботі описано кілька нейронних мереж, де зворотне розповсюдження працює набагато швидше, ніж попередні підходи до навчання, що робить можливим використання нейронних мереж для вирішення проблем, які вирішити раніше було неможливо.[2] Сьогодні алгоритм зворотного розповсюдження є основною для навчання нейронних мереж.

Алгоритм зворотного розповсюдження похибки є контрольованим методом навчання для багатошарових мереж зворотного зв'язку із галузі штучних нейронних мереж.

Нейронні мережі зворотного зв'язку працюють завдяки принципу обробки інформації однієї або декількох нейронних клітин, які мають назву нейрони. Нейрон приймає вхідні сигнали через свої дендрити, які передають електричний сигнал до тіла клітини. Аксон передає сигнал до синапсів, які є з'єднаннями аксона клітини з дендритами інших клітин.[1] Як видно з опису основних складових, мережа дуже сильно нагадує структуру людського мозку з нейронами, синапсами, дендритами та аксонами.

Сам принцип підходу зворотного розповсюдження полягає в моделюванні заданої функції шляхом постійної модифікації внутрішніх ваг вхідних сигналів для отримання бажаного вихідного сигналу. Система тренується за допомогою контрольованого методу навчання, де похибка між результатом

роботи системи та відомим очікуваним результатом надається системі та використовується для подальшої модифікації її внутрішніх ваг.

По суті, алгоритм зворотного розповсюдження є методом тренування ваг у багатошаровій нейронній мережі прямого-зворотного зв'язку. Алгоритм вимагає визначення мережевої структури з одного або декількох шарів, де всі шари повністю пов’язані одне з одним. Стандартна мережева структура - це один вхідний рівень, один прихований рівень і один вихідний рівень, як можна побачити на Рисунку 1.1. Але ця структура також не забороняє створення додаткових шарів, наприклад, цілком можлива структура з двома шарами прихованого рівня.

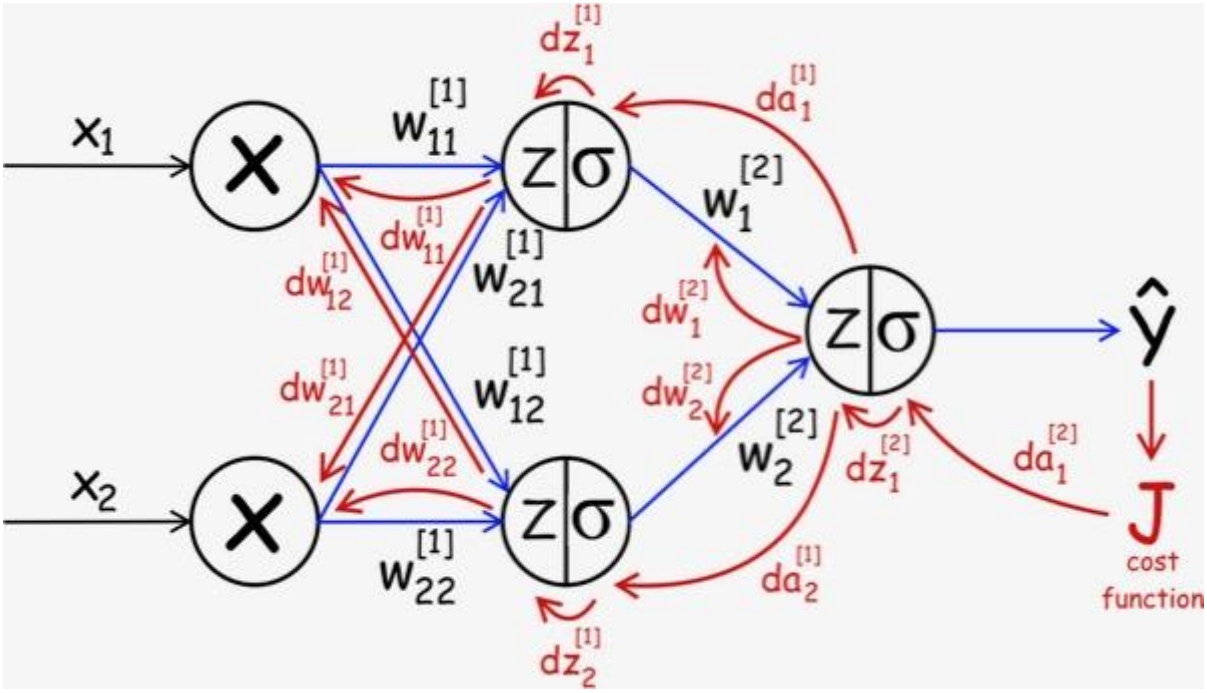

Рисунок 1.1 Нейронна структура на основі зворотного поширення похибки.[3]

Також необхідно розібратись з деякими визначеннями, які будуть зустрічатись далі у розділі.

“A **dataset** is a set of data that is collected for a specific purpose. There are many ways in which data can be collected—for example, as part of service delivery, one-off surveys, interviews, observations, and so on. In order to ensure that the

meaning of data in the data set is clearly understood and data can be consistently collected and used, data are defined using metadata…"[a]

"The idea of **weight** is a foundational concept in artificial neural networks. A set of weighted inputs allows each artificial neuron or node in the system to produce related outputs. Professionals dealing with machine learning and artificial intelligence projects where artificial neural networks for similar systems are used often talk about weight as a function of both biological and technological systems.

Weight is also known as **synaptic weight**."[b]

[a] "A guide to data development", National Data Development and Standards Unit in Australia, 2007
[b] Technopedia, What does Weight mean?
https://www.techopedia.com/definition/33274/weight-neural-networks

Ініціалізація шарів нейронної мережі

Реалізація алгоритму починається з створення самої нейронної мережі. Для цього необхідно створити як мінімум 3 шари нейронів, які будуть пов’язані між собою. Це дозволить приймати на вхід дані, обраховувати результат та змінювати ваги в нейронах, якщо результат не відповідає потрібному.

Кожен нейрон насправді має не одну вагу, а дві, одна вага для кожного вхідного з'єднання та додаткова вага для зміщення. Для зберігання ваг доречно використовувати словник, тому для цього варто використовувати словник для представлення кожного нейрона та зберігати властивості за іменами.

Кількість нейронів на вхідному рівні дорівнює кількості вхідних даних в датасеті, кількість нейронів на прихованому рівні може варіюватись і зазвичай треба тестувати точність системи залежно від різної кількості нейронів на прихованому рівні, далі вже обирати ту кількість, яка демонструє найкращі показники. Кількість нейронів на вихідному рівні в більшості випадків один, так як результат зазвичай одне число.

Також хорошою практикою є ініціалізація ваг якимось невеликим числом, наприклад, від нуля до одиниці.

Поширення вперед

Для обрахування вихідних даних за допомогою нейронної мережі, необхідно поширити вхідний сигнал через кожен шар, допоки вихідний рівень не виведе свої значення. Цей процес має назву *поширення вперед*.

Поширення вперед необхідно для того, щоб робити прогнози щодо результуючого значення, яке в подальшому треба буде виправляти (збільшувати або зменшувати) для правильного отримання результату.

Умовно процес поширення вперед можна розділити на 3 етапи:

1. Активація нейронів.
2. Передача нейронів.
3. Поширення вперед

Першим кроком є обрахунок активації одного нейрона, який отримує вхідні дані. У випадку з прихованим рівнем це може бути один рядок з навчального датасету. Також даними можуть бути результат з кожного нейрона на прихованому шарі, якщо обрахунок активації ведеться на вихідному шарі.

Активація обраховується, як зважена сума вхідних даних. Дуже подібно до лінійної регресії.

Далі йде передача нейронів. Як тільки нейрон активований, необхідно передати активацію, щоб побачити результат. Для передачі нейронів можна використовувати різні функції. Традиційно у ролі активуючої функції використовується сигмоїда, вона має вигляд латинської букви «S», *див Рисунок 1.2.* Також сигмоїду ще називають «логістичною функцією». Сигмоїда приймає будь-яке значення та генерує число між нулем та одиницею. Більш того, розрахувати похідну сигмоїди досить легко, це дуже спрощує наступні етапи.

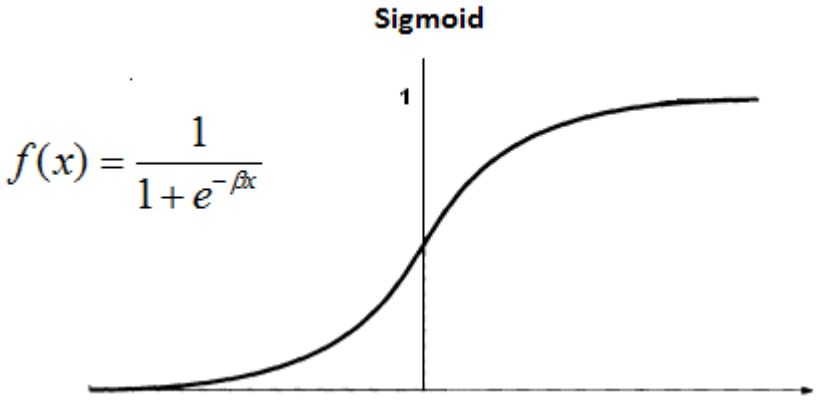

Рисунок 1.2 Вигляд графіку Сигмоїди.[4]

Однак в якості функції передачі можна використовувати наприклад Tanh (гіперболічний тангенс). Також зовсім нещодавно «випрямляюча» функція передачі продемонструвала свою ефективність у великих нейронних мережах глибокого навчання.

І нарешті сам етап поширення уперед. Цей етап є досить прямолінійний, необхідно лише обробити кожен нейронної мережі вираховуючи результат для кожного нейрона. Всі виходи одного шару перетворюються у входи наступного шару.

Для кожного нейрону спочатку активуємо його, потім передаємо і результат записуємо в певний масив який стає входами для масиву наступного шару.

Зворотне поширення похибки

Алгоритм зворотного поширення похибки отримав назву саме через цей етап і через спосіб, яким тренуються ваги в нейронах.

Похибка обраховується між очікуваними входами та виходами, які поширюється вперед по мережі. Далі отримані похибки поширюються назад від вихідного шару до прихованого шару, на шляху призначається «винуватець» похибки та оновлюються ваги.

Цей етап умовно можна розділити на 2 підетапи:

1. Передача похідної.
2. Зворотне поширення похибки.

Спочатку необхідно обрахувати нахил вихідного значення з нейрону. Нахил обраховується за допомогою похідної, як було вже зазначено раніше похідна від сигмоїди обраховується досить легко. Похідна виглядає наступним чином:

*output * (1.0 - output)*

Де *output* – вихідне значення нейрона.

Це дасть сигнал про помилку який далі зворотно поширюється через мережу. Похибка обраховується як різниця між очікуваним значенням і отриманим помножену на результат функції обрахунку похідної сигмоїди. Це обчислення похибки використовується нейронами на вихідному шарі. Очікуване значення – це значення самого класу. Однак на прихованому рівні все відбувається трохи складніше.

Сигнал похибки для прихованого шару обраховується як зважена помилка для кожного нейрону на вихідному шарі. Тепер можливо застосувати зворотне поширення похибки для того щоб натренувати нейронну мережу.

Тренування мережі

В процесі тренування мережі головною задачею є мінімізувати похибку в прогнозуванні. Тренування відбувається за допомогою стохастичного градієнтного спадання. На початку похибка є великою, що не дозволяє мережі робити точні прогнозування, але з кожною епохою похибка стає все менше, а ваги в мережі підлаштовуються під вимоги користувача мережею, див Рисунок 1.3.

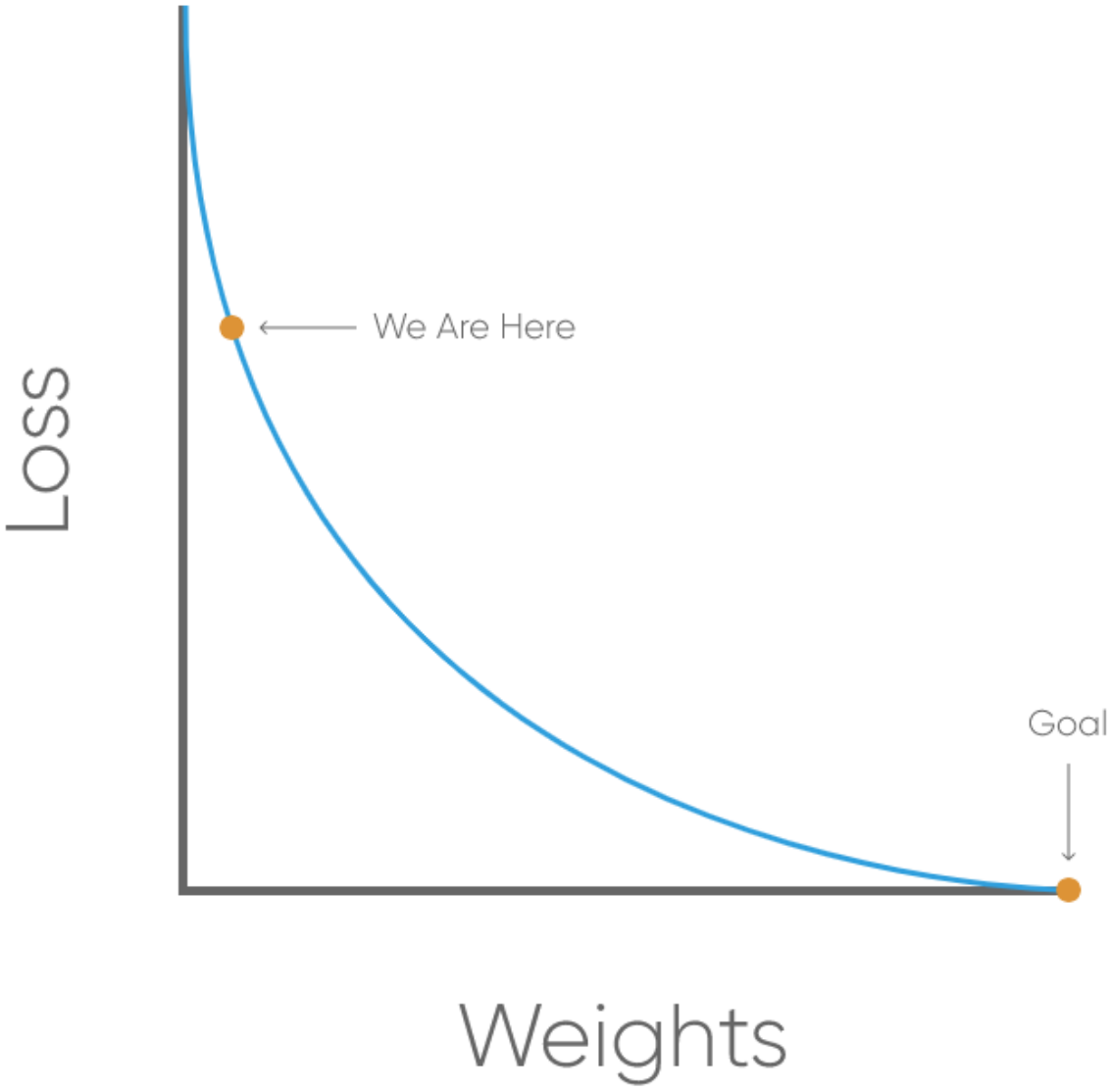

Рисунок 1.3 Зменшення похибки і покращення ваг.[4]

Тренування мережі включає в себе викриття датасету для кожного рядка, що поширюються вперед мережею з подальшим зворотним поширенням похибки і оновленням ваг мережі.

Спочатку необхідно визначити функцію, що буде рахувати нове значення для ваг. Вага розраховується наступним чином:

*вага = вага + швидкість_навчання * похибка * вхідне_значення*

швидкість_навчання - це параметр, який задається користувачем системи, чим менше значення, тим повільніше буде тренування мережі, але тим точніше буде прогнозування після тренування.

Для прикладу, якщо значення швидкості навчання буде дорівнювати нуль цілим одній десятій вага оновиться на десять відсотків від кількості, яка теоретично могла б оновитись. Як правило необхідно ставити низьку швидкість навчання, для того, щоб система могла підібрати хороший набір вагових коефіцієнтів з більшою ймовірністю. Якщо ж поставити велику швидкість навчання система прийде до найшвидшого набору коефіцієнтів, що мінімізує помилки. Це ще зветься передчасною конвергенцією.

Тепер необхідно провести сам процес тренування. Як вже зазначалось раніше мережа оновлюється за допомогою стохастичного градієнтного спадання. Сама функція тренування передбачає, що користувач системою має спочатку визначити кількість епох тренування, в залежності від цієї кількості буде варіюватись кількість кіл тренувань. Кожне коло тренування мережа оновлюється для кожного рядка в датасеті.

Навчання мережі може відбуватись одним з двох способів: оновлення відбуваються для кожної моделі тренування, такий тип навчання має назву *онлайн навчання* або коли помилки накопичуються протягом епохи до оновлення ваг, таке навчання має назву *пакетне навчання* або *пакетним градієнтним спуском*.

“In **batch learning** the machine learning model is trained using the entire dataset that is available at a certain point in time. Once we have a model that performs well on the test set, the model is shipped for production and thus learning ends. This process is also called **offline learning**.”[c]

“**Online machine learning** is a method of machine learning in which data becomes available in a sequential order and is used to update the best predictor for future data at each step, as opposed to batch learning techniques which generate the best predictor by learning on the entire training data set at once”[d]

[c] Building Machine Learning and Deep Learning Models on Google Cloud Platform, Ekaba Bisong, 28 Вересня 2019, сторінки 199-201

[d] Wikipedia, Online Machine Learning, https://en.wikipedia.org/wiki/Online_machine_learning#:~:text=In%20computer%20science%2C%20online%20machine,the%20entire%20training%20data%20set

Прогнозування

Скласти прогнозування за допомогою вже до того навченої мережі досить легко. До того вже було описано, як поширювати вхідне значення вперед для того, щоб отримати вихідне значення. І насправді це все що потрібно, щоб зробити прогноз. Можна використовувати самі вихідні значення напряму у ролі ймовірності шаблону, що належить кожному класу виводу.

Також може бути корисним повернути результат назад у чітке передбачення класу. Це можна зробити обравши значення класу, яке має найбільшу ймовірність. Такий вибір також має назву *аргумент максимізації.*

“Аргумент максимізації (argmax або arg max) — значення аргументу, при якому даний вираз досягає максимуму. Іншими словами, argmax *f(x)* - є значення *x* при якому *f(x)* досягає свого найбільшого значення. Є розв'язком задачі максимізації функції скінченної кількості аргументів ”[e]

Тож які проблеми може вирішувати мережа на основі зворотного поширення похибки? Зворотне поширення похибки чудово підходить для проблеми класифікації.

У машинному навчанні класифікація - це концепція контрольованого навчання, яка полягає у розподіленні вхідних даних по певним класам. Серед найпоширеніших проблем класифікації є проблеми по типу розпізнавання мови, розпізнавання обличчя, класифікація документів, розпізнавання об’єктів на зображеннях. Це можуть бути як проблеми бінарної класифікації, коли вибір допустимих значень обмежений тільки «істиною» та «хибою» або нулем та одиницею, а також проблеми багатокласового характеру.[5]

Для таких проблем алгоритм зворотного поширення похибки працює просто неперевершено, з достатнім рівнем натренованості, мережа без усіляких

[e] Виноградов І. М. Математична енциклопедія. - Радянська енциклопедія, 1982 рік - 1 184 с.

проблем досягає точності у класифікації до 98%. Особливо це добре працює у випадку бінарної класифікації.

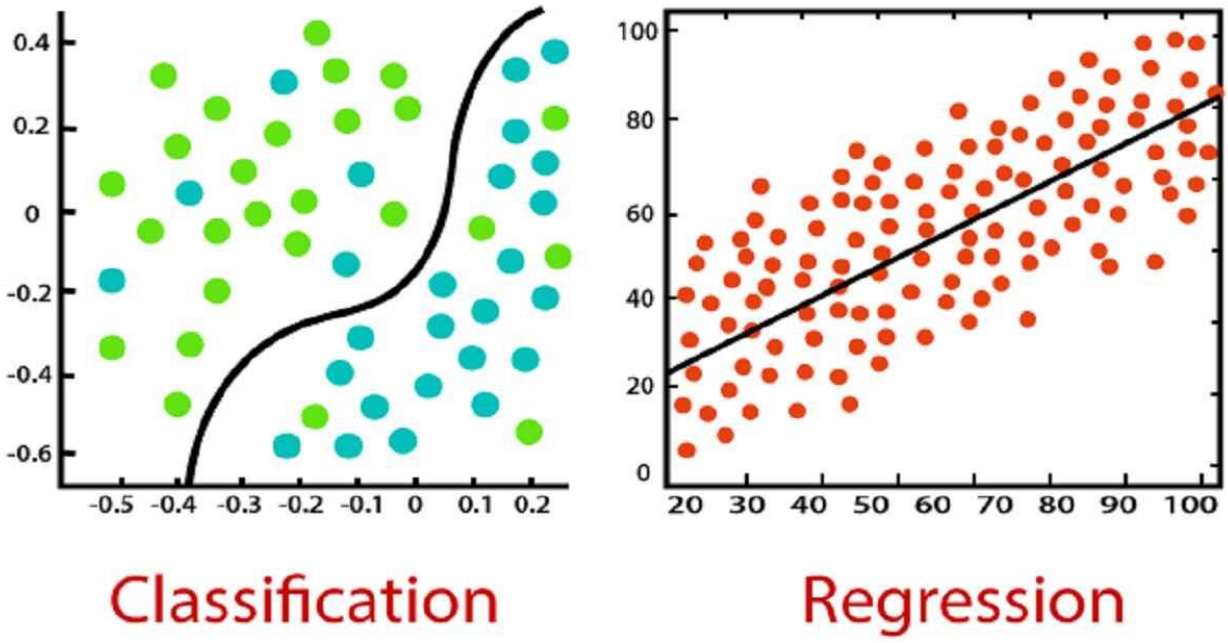

Рисунок 1.4 Задачі класифікації та регресії

Що ж на рахунок регресії і прогнозування часових рядів? Тут все набагато складніше. Для прогнозування конкретного числа необхідно багато епох тренування і все рівно результати мережі можуть відхилятись від потрібного результату на малі значення, але якщо використовувати результати мережі для подальшого прогнозування в якості вхідних даних на довгій дистанції ця похибка у прогнозуванні все збільшується, що не дозволяє отримати чіткого прогнозування.

Також є певні проблеми з якими може зіштовхнутись мережа на основі алгоритму зворотного поширення похибки. Однією з таких проблем є *проблема зникання градієнту*.

Проблема зникання градієнту виникає під час тренування штучних нейронних мереж на основі стохастичного градієнту та зворотного поширення похибки. Певні активаційні функції, наприклад сигмоїда, стискає великий вхідний простір у малий між нулем та одиницею. Через це велика

зміна вводу сигмоїдної функції спричинить невелику зміну виводу, це в свою чергу спричиняє те, що похідна стає малою.[6]

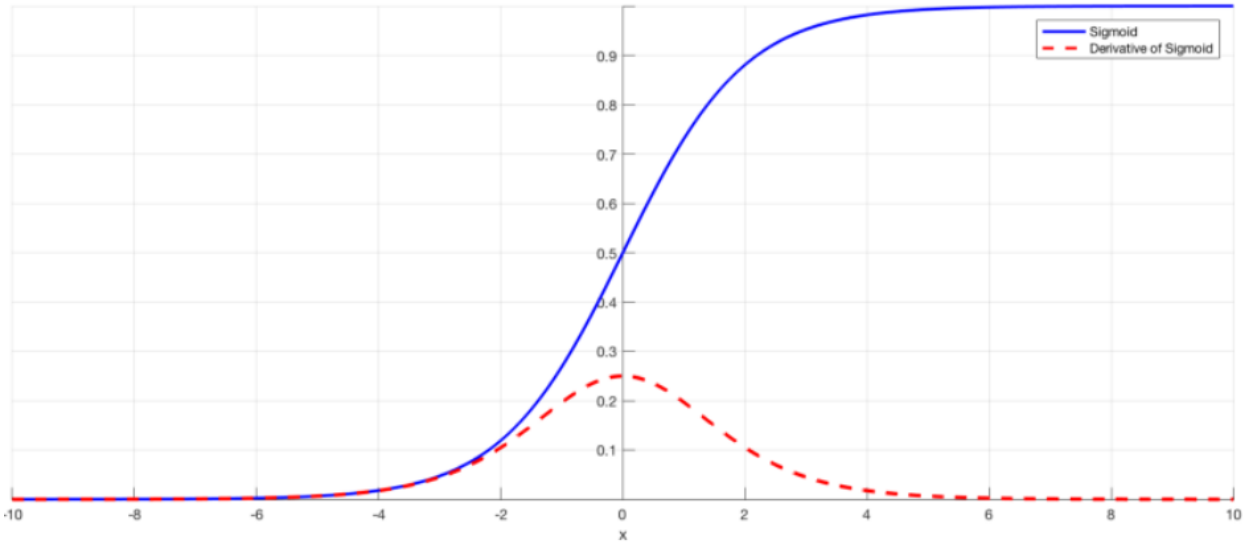

Рисунок 1.5 Сигмоїда і її похідна.

На рисунку 1.5 видно, що коли входи сигмоїдної функції стають більшими або меншими (коли абсолютне значення *x* стає більшим), значення похідної стає близьким до нуля.

Це призводить до того, що ваги в мережі перестають змінюватись, або змінюються дуже повільно. В найгірших сценаріях, це повністю зупиняє нейронну мережу від подальшого навчання. Також у випадках використання активуючої функції, похідні якої набувають великих значень є ризик зіткнутись з *проблемою вибуху градієнту*. Коли система перестає навчатись з протилежних причин.

Як же уникнути всіх цих проблем пов’язаних з алгоритмом зворотного поширення похибки? Найпростіше рішення – використати нейронну мережу, в основі якої закладений кращий алгоритм.
Тут у гру вступає мережа з довгою короткочасною пам’яттю.

Архітектура нейронних мереж з довгою короткочасною пам’яттю.

Рекурентні нейронні мережі

Для того, щоб зрозуміти, як працює мережа на основі довгої короткочасною пам’яттю спочатку необхідно зрозуміти, як працюють *Рекурентні нейронні мережі*, так як мережі з довгою короткочасною пам’яттю це особливий вид РНМ.

РНМ – це тип нейронної мережі, коли вихідні дані попереднього кроку подаються як дані на вхід для поточного кроку. Перефразовуючи, рекурентна нейронна мережа, являє собою узагальнення нейронної мережі прямого зв’язку, де є внутрішня пам’ять. РНМ використовується для розпізнавання послідовних характеристик даних та використання певних шаблонів для прогнозування[7], з останнього стає чітко зрозуміло, що рекурентні нейронні мережі це якраз те що необхідно для прогнозування часових рядів.

Передбачається, що в звичайній нейронній мережі всі входи та виходи незалежні одне від одного. Однак такий підхід не спрацює в ситуаціях коли треба прогнозувати наступне слово в реченні, або наступне число в послідовності чисел, бо це буде значно краще, якщо буде відомо яке число/слово йшло до того.

Ідея рекурентної нейронної мережі полягає у використанні послідовної інформації, вони називаються рекурентними, оскільки виконують однаковий процес для кожного елемента послідовності, при цьому вихідні дані залежать від попередніх обчислень, також варто зазначити, що рекурентна нейронна мережа має певну “пам’ять”, яка фіксує інформацію, що було нараховано до цього часу.

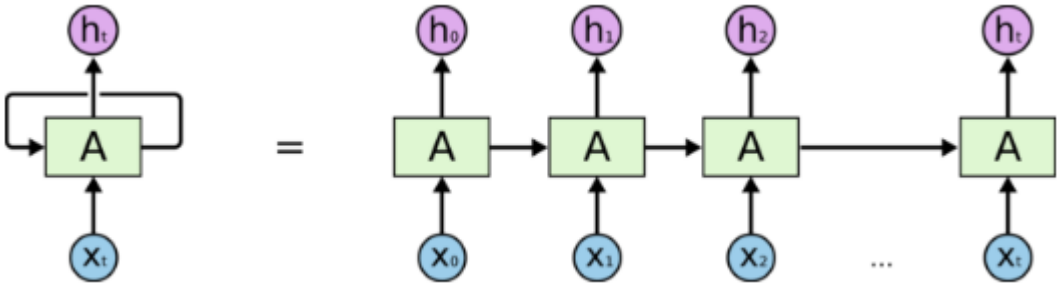

Рисунок 2.1 Архітектура рекурентної нейронної мережі

Довга короткочасна пам’ять

РНМ має проблеми пов’язані з короткочасною пам’яттю. У випадку, якщо послідовність досить довга, рекурентним нейронним мережам важко переносити інформацію з попередніх часових етапів на пізніші. Тож якщо на маті стоїть обробка цілого параграфу тексту, РНМ може упустити важливу інформацію з початку параграфа.

З цих причин оптимальним рішенням є застосування довгої короткочасної пам’яті, спеціальний вид рекурентних нейронних мереж здатна до запам’ятовування довгострокових залежностей. Іншими словами, нейронна мережа на основі довгої короткочасної пам’яті здатна запам’ятовувати інформація на тривалий період часу.

Більш того, мережа на основі довгої короткочасної пам’яті повністю унеможливлює виникнення проблеми зникання градієнту, завдяки використання покращеної активуючої функції.

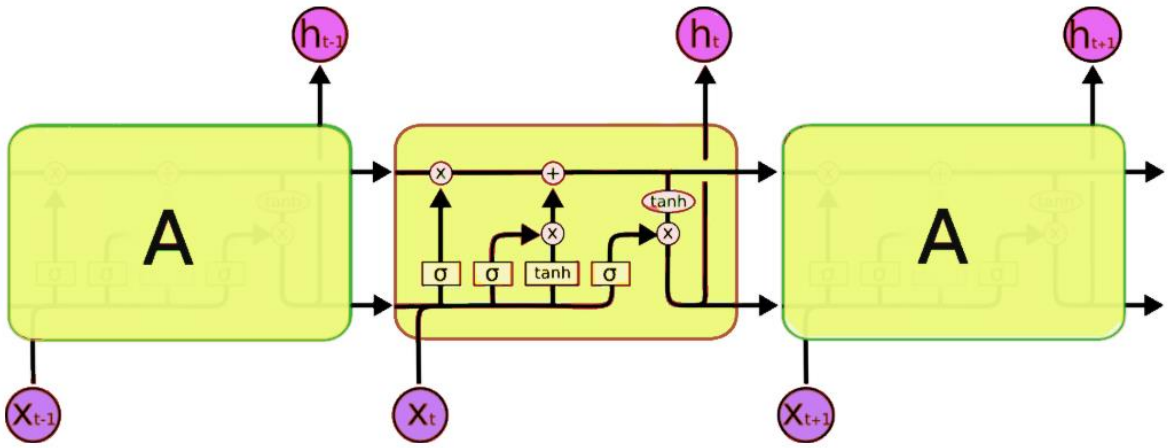

Рисунок 2.2 Архітектура мережі короткочасної довготривалої пам’яті

Короткочасна довготривала пам’ять має триступеневий процес, що складається з шлюзу забуття, вхідного шлюзу та вихідного шлюзу.

Шлюзи:

- Шлюзи це сигмоїдний шар нейронної мережі, за яким йде точковий оператор множення.
- Шлюзи контролюють потік інформації з та до пам’яті.
- Шлюзами керує конкатенація виходу з попереднього кроку і поточного входу, а також вектор стану комірки.

Шлюз забуття:

- Контролює, яку інформацію потрібно викинути з пам’яті або іншими словами, забути.
- Вирішує яку частину минулого потрібно зберігати в пам’яті.

Вхідний шлюз (Шлюз оновлення):

- Керує тим, яка інформація має додатись до стану комірки з поточного вводу.
- Вирішує скільки цієї одиниці інформації буде додано до поточного стану.

Вихідний шлюз:

- Завдяки певним умовам, вирішує, що потрібно виводити з пам’яті.
- Вирішує, яка саме частина поточної комірки потрапляє на вихід.

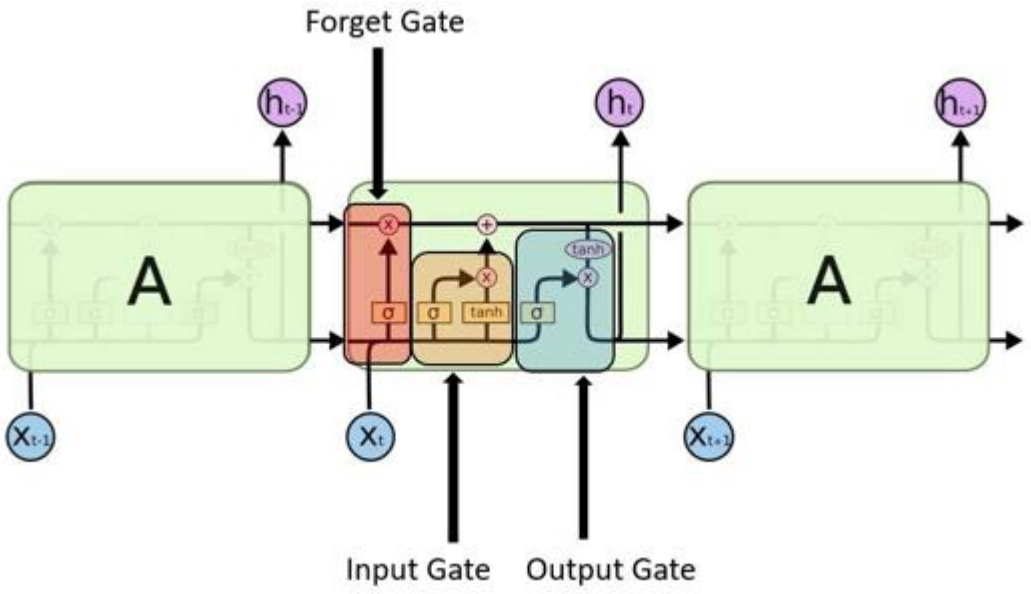

Рисунок 2.3 Триступеневий процес з трьома рівнями шлюзів

Ще одним із основних компонентів мережі з довгостроковою пам’яттю є комірка пам’яті, вона може підтримувати свій стан протягом часу. Комірка складається з явної пам’яті (також називають вектором стану комірки) та блоків керування. Блоки керування регулюють вхідний потік інформації в пам’ять і також з пам’яті.

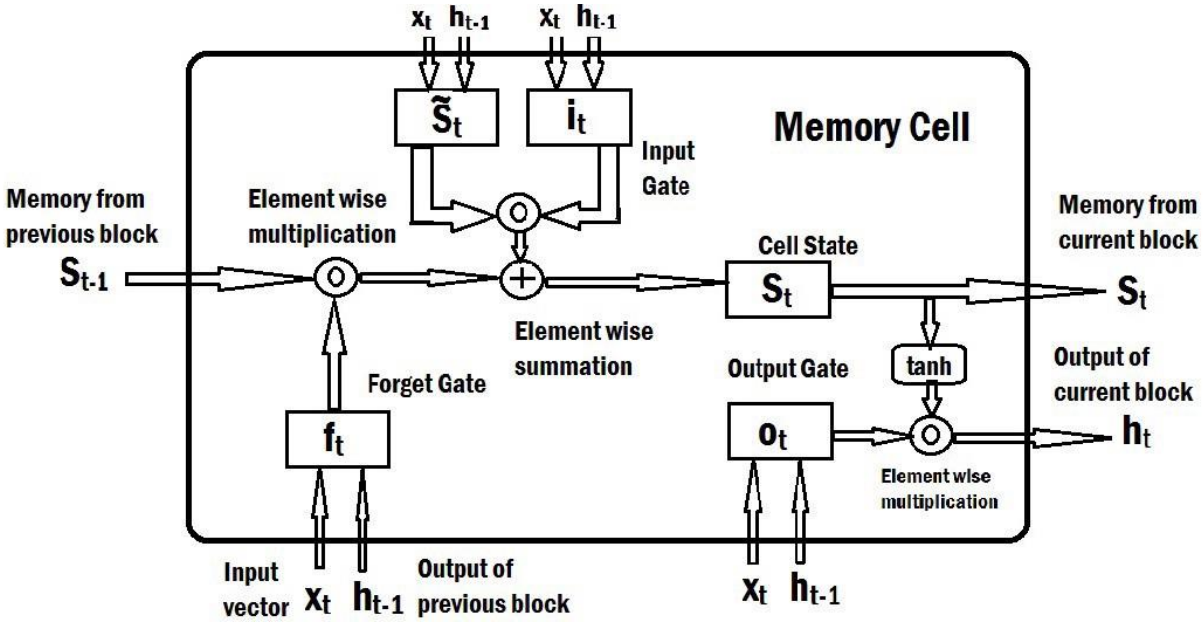

Рисунок 2.4 Структура комірки в мережі

Результати з прогнозування часових рядів

Опис обраних даних

У якості даних для тренування нейронної мережі, а також як тестових даних для подальшого прогнозування було обрано датасет, який містить дані про кількість пасажирів на борту літаків.

У самому датасеті дані розділені на місяці, починаючи з 1949 року, структуру датасету видно на цьому скріншоті, див. Рисунок 3.1.

Рисунок 3.1 Вигляд даних про кількість пасажирів

Не важко здогадатись, що кількість пасажирів з кожним роком буде тільки збільшуватись порівняно з попереднім роком, так як в двадцятому столітті польоти літаком тільки набирали популярність. Якщо подивитись на графік датасету, чудово видно, як графік має певне зростання схоже на логарифмічне, див Рисунок 3.2.

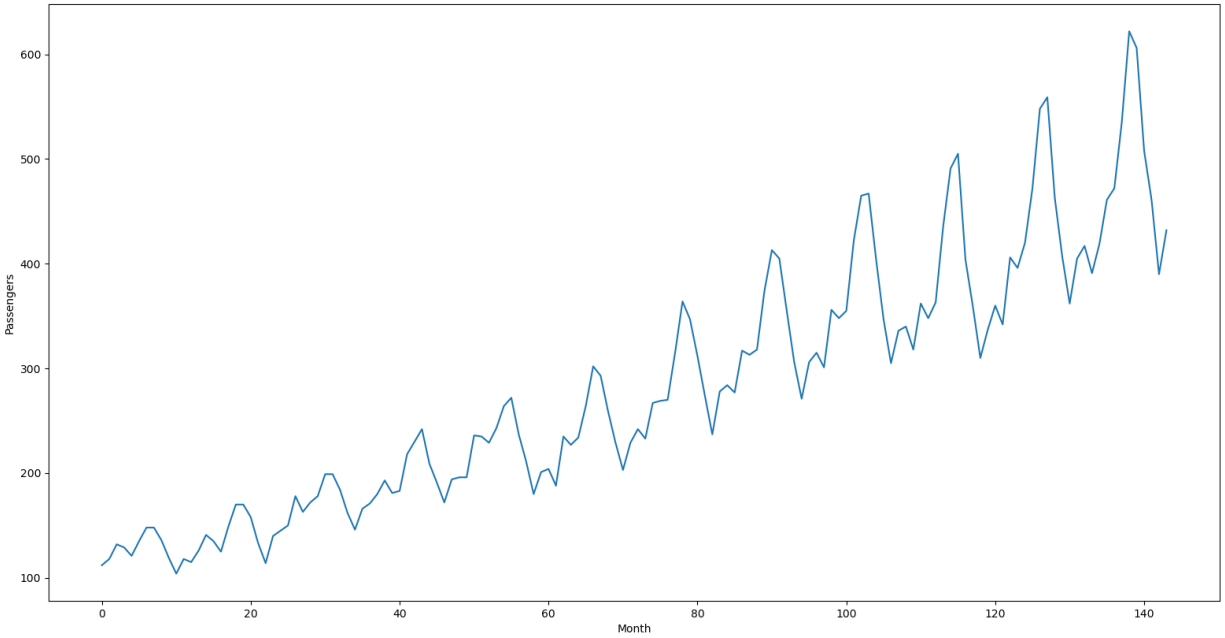

Рисунок 3.2 Графік датасету кількості пасажирів

Також дуже гарно видно певний шаблон графіку, спочатку року кількість пасажирів завжди менше ніж в кінці року, гадаю, що це пов’язано з новорічними святами, часом коли більшість людей їде подорожувати, і у випадках коли ці подорожі на далекі дистанції, користуються літаками.

Більш того, дуже чітко проглядається сезонність датасету. В термінології часових рядів, сезонність – це наявність змін, які трапляються через певні проміжки часу, наприклад, щотижня, щомісяця, щоквартально. Сезонність складається з періодичних, повторюваних та загалом регулярних та передбачуваних закономірностей на рівні часового ряду.

В даному випадку кожен рік майже ідентичний попередньому року і це дуже чітко видно з графіку.

Опис роботи програми

Програма працює на середовищі Tensorflow.Keras, які були встановлені за допомогою дистрибутива Anaconda, головна ціль якого полегшити процес управління та розгортання пакетів пов'язаних з нейронними мережами. З пакету Tensorflow необхідно інмортувати модель Sequential, а також шари LSTM та Dense, див. Рисунок 3.3.

```
from tensorflow.python.keras.layers import LSTM, Dense
from tensorflow.python.keras.models import Sequential
```

Рисунок 3.3 Підключення потрібних модулів з Tensorflow.Keras

Також для зручного масштабування даних та використання функції середньої квадратичної помилки було підключено бібліотеку sklearn, див. Рисунок 3.4.

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import mean_squared_error
```

Рисунок 3.4 Підключення потрібних модулів з sklearn

У статистиці середньоквадратична помилка (MSE) вимірює середнє значення квадратів помилок, тобто середня квадратична різниця між розрахунковими значеннями та фактичною величиною. MSE - це функція ризику, яка відповідає очікуваному значенню квадратних втрат помилок.[8]

Сама робота програми складається з таких етапів:

- Зчитування даних з csv файлу за допомогою бібліотеки pandas. Додаток А.
- Масштабування даних до проміжку від нуля до одного. Додаток Б.

- Розділення датасету на 2 частини: тренувальну та тестову. В даному випадку датасет розділяється навпіл. Додаток В.
- Приведення даних до вигляду, де X - кількість пасажирів на певний час (t), а Y - кількість пасажирів на наступний час (t + 1). Додаток Г.
- Змінення форми датасету за допомогою numpy.reshape(). Додаток Ґ.
- Створення та підготовка нейронної мережі з довгою короткочасною пам’яттю. Додаток Д.
- Створення готового прогнозування для тренувальної частини та тестової. Додаток Е.
- Зворотна трансформація отриманих даних з проміжку від [0, 1] до реальних даних. Додаток Є.
- Зміщення результатів прогнозування на певне значення, задля усунення накладання графіків. Додаток Ж.
- Побудова графіка результатів за допомогою бібліотеки matplotlib.pyplot. Додаток З.

Це і є основні етапи в роботі програми. Сам код програми вийшов досить коротким, через те що багато функціоналу, який потрібен для прогнозування вже вбудований в бібліотеки, які використовуються в роботі.

Спочатку була ідея зробити все без використання сторонніх бібліотек, але це в результаті зайняло дуже багато часу і досягти необхідного результату так і не вийшло. Звісно ці всі операції можна зробити майже вручну, без застосування бібліотек sklearn, tensorflow, numpy та інших. Але такий підхід є дуже не оптимальним та нерозумним.

Отримані результати

Результати прогнозування напряму залежать від кількості епох тренування нейронної мережі. Розберемо деякі результаті в залежності від кількості епох.

При малій кількості епох, наприклад, десять, очевидно, що результати прогнозування будуть дуже неточними і це підтверджує графік, рисунок 3.5.

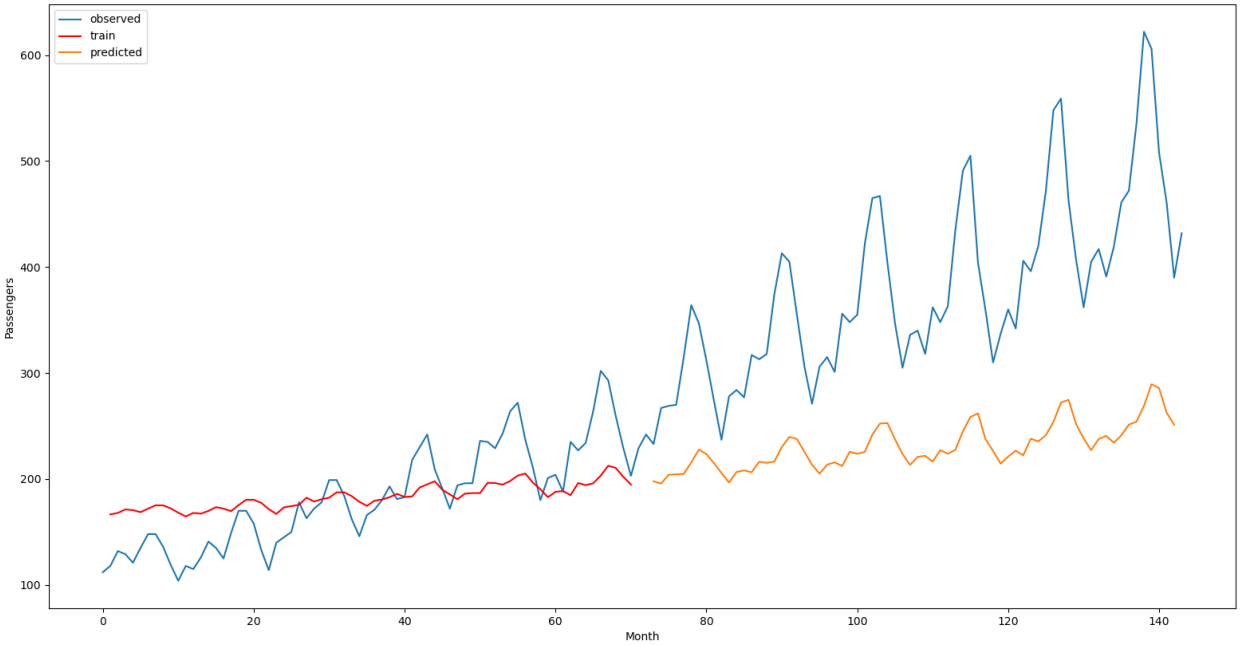

Рисунок 3.5 Результати при кількості епох рівній десяти

Як видно з графіку, при такій малій кількості епох тренування результати дуже далекі від потрібних. Також можна подивитись на результати виводу під час тренування.

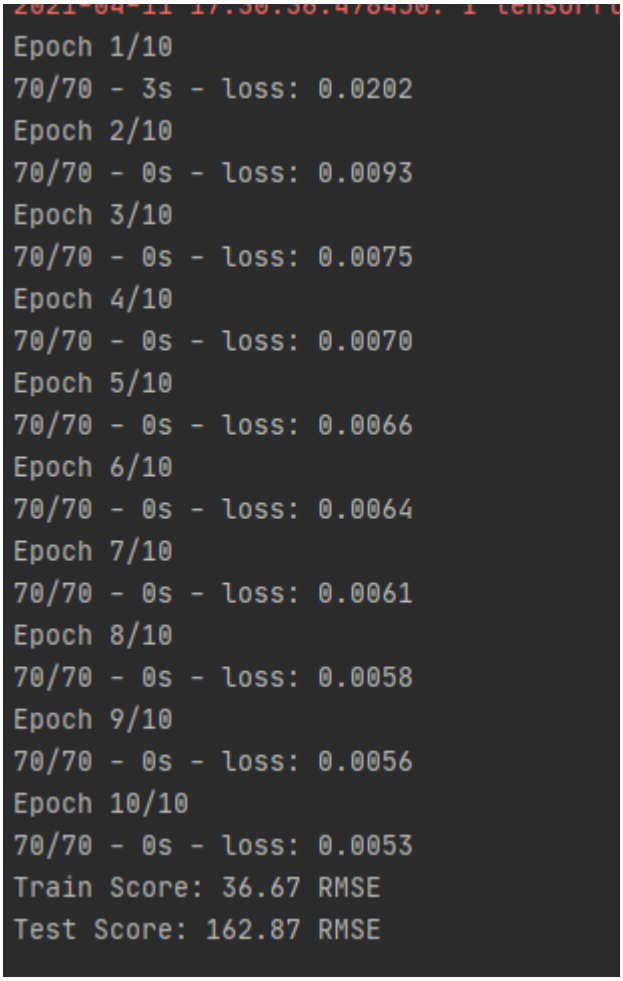

Рисунок 3.6 Вивід тренування при кількості епох рівній десяти

Тут можна побачити яке значення має похибка в системі (на останній ітерації похибка дорівнювала 0.0053). Виводиться кількість епох, які вже пройшли і скільки епох ще буде пройдено. В кінці підводиться підсумок по середньоквадратичній похибці після тренування для обох датасетів.

При кількості епох рівній п’ятдесяти, похибка вже перестає зменшуватись і результати прогнозування стають дуже чіткими, це дуже гарно видно з графіку, Рисунок 3.7

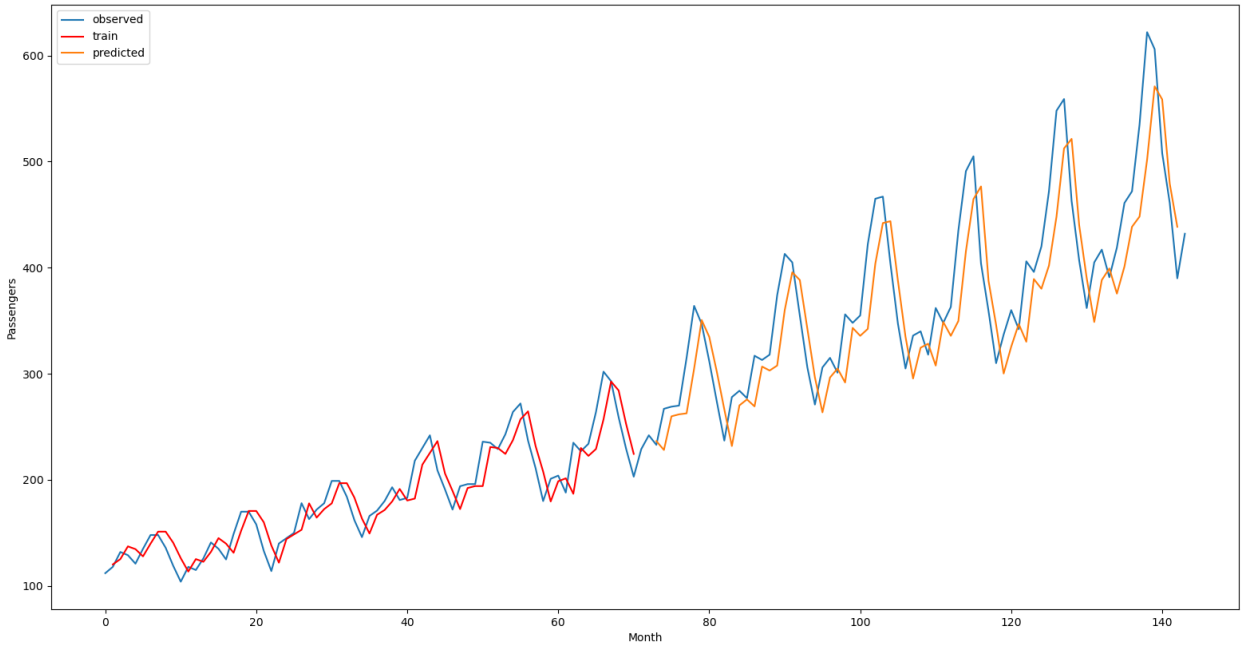

Рисунок 3.7 Результати прогнозування при кількості епох рівній п'ятдесяти

Також варто зазначити що графік спеціально зміщений праворуч та вниз на певне значення, яке задається вручну, тому прогнозування насправді є дуже точним і отримані значення повністю співпадають з значеннями спостерігаємого датасету.

Значення похибки під час епох і значення середньоквадратичної похибки в кінці тренування видно на рисунку 3.8.

```
Epoch 47/50
70/70 - 0s - loss: 0.0013
Epoch 48/50
70/70 - 0s - loss: 0.0013
Epoch 49/50
70/70 - 0s - loss: 0.0013
Epoch 50/50
70/70 - 0s - loss: 0.0013
Train Score: 18.58 RMSE
Test Score: 45.65 RMSE
```

Рисунок 3.8 Вивід тренування при кількості епох рівній п'ятдесяти

Як видно з рисунку на останніх трьох ітераціях похибка вже не змінювалась, що говорить про достатній натренованості нейронної мережі.

Порівняння з іншими способами прогнозування

Опис методу ARIMA

Задля порівняння я також використав метод ARIMA, що є дуже поширеним для прогнозування часових рядів.

ARIMA – це скорочення від Auto Regressive Integrated Moving Average (інтегрована модель авторегресії змінного середнього). Насправді це є клас, який певним чином “пояснює” даний йому часовий ряд на основі власних минулих значень, власних відставань та відсталих помилок прогнозування.

Будь-який часовий ряд, який демонструє певну шаблонність та не є випадковим білим шумом, можливо змоделювати на моделях ARIMA. Також важливим критерієм є не сезонність часового ряду. Як вже було зазначено раніше датасет, що використовується для прикладу прогнозування є сезонним в таких випадках використовується SARIMA скорочення від “Seasonal ARIMA”.[9]

SARIMAX використовується на наборах даних, які мають сезонні цикли. Різниця між ARIMA та SARIMAX полягає у сезонності та екзогенних факторах (сезонність та звичайна ARIMA погано поєднуються).

SARIMAX э досить складною, але ключовим фактором є те, що для SARIMAX потрібні не лише аргументи p, d та q, які потрібні ARIMA, але також потрібен інший набір аргументів p, d та q для сезонного аспекту, а також аргумент під назвою “s”, який є періодичністю сезонного циклу даних.

Вибираючи значення s, потрібно брати до уваги уявлення про те, які в системі циклічні сезонні дані. Якщо точки даних розділені щомісячно, а сезонний цикл - рік, то необхідно встановити s на 12. Або якщо точки даних

розділені щоденно, а сезонний цикл - тиждень, тоді s потрібно встановити рівним 7.

Порівняння результатів

Насправді на датасеті з кількістю пасажирів результати вийшли ідентичні, але якщо робити детальне дослідження, яке присвячене порівнянню різних методів прогнозування, на різних формах даних ці методи прогнозування показують дещо різні результати, таке детальне порівняння не буде розглянуто в цій курсовій роботі, бо це заслуговує окремої курсової роботи.

В більшості випадків метод ARIMA все ж таки демонструє трохи кращі результати ніж нейронні мережі на основі довгої короткочасної пам’яті. Можливо, необхідно застосовувати біль потужні архітектури нейронних мереж, для перевершення результатів прогнозування з використанням ARIMA, наприклад, мережі з глибоким навчанням або мультишарових мережах.

Однак варто зазначити деякі особливості ARIMA, по-перше, робота з методом ARIMA є значно простою: мережу не потрібно тренувати багато ітерацій для досягнення потрібних результатів, дані можна вносити в модель без попереднього трансформування до вигляду від одного до одиниці.

По-друге, в бібліотеці statsmodels.api є можливість автоматичного визначення найкращих термів в моделі ARIMA завдяки функції auto_arima, це значно спрощує роботу над прогнозуванням.

Висновки

У цій курсовій роботі було описано роботу алгоритму зворотного поширення похибки, який є основою більшості сучасних нейронних мереж, розглянуті всі етапи цього алгоритму та його особливості. Описано завдяки чому нейронна мережа в основі якої лежить алгоритм зворотного поширення помилки може самостійно навчатись та видавати певні результати. Розглянутий процес активації мережі, поширення вперед та зворотного поширення помилки, що є головними складовими всього алгоритму.

В кінці першого розділу описані проблемо та недоліки “сирого” так би мовити алгоритму зворотного поширення помилки, для яких задач він погано підходить та які проблеми можуть спіткати в процесі спроб вирішення задач за допомогою цього алгоритму.

Далі описано спеціальний вид нейронних мереж під назвою – рекурентні нейронні мережі, їх особливості та використання, а також описані особливості роботи мережі з довгою короткостроковою пам’яттю, яка і була застосована для прогнозування.

В розділі, який присвячено роботі програми описані тестові дані, які були взяті для прогнозування та коротко розказано з яких етапів складається програма. Також коротко було описано інший спосіб прогнозування, відмінний від того, що використовувався в курсовій роботі та його особливості

Загалом вважаю, що мета курсової роботи була досягнута, звісно на даний момент самих тестів з прогнозування проведено досить мало. В перспективі необхідно розширити програму, додавши більшу кількість різноманітних даних для прогнозування, також можна розглянути і інші нейронні мережі та інші способи прогнозування, окрім ARIMA.

Список літератури

1. How to Code a Neural Network with Backpropagation In Python (from scratch). Від Jason Brownlee 7 листопада, 2016. https://machinelearningmastery.com/implement-backpropagation-algorithm-scratch-python/
2. Michael A. Nielsen, “Neural Networks and Deep Learning”, Determination Press, 2015 http://neuralnetworksanddeeplearning.com/chap2.html
3. Understand and Implement the Backpropagation Algorithm From Scratch In Python, Abhisek Jana, 15 квітня, 2019 http://www.adeveloperdiary.com/data-science/machine-learning/understand-and-implement-the-backpropagation-algorithm-from-scratch-in-python/
4. Build a Neural Network, Samay Shandasani, 2017 https://tryenlight.github.io/neural-network
5. Edureka, classification in machine learning, 17 липня 2020 https://www.edureka.co/blog/classification-in-machine-learning/#:~:text=In%20machine%20learning%2C%20classification%20is,recognition%2C%20document%20classification%2C%20etc.
6. Towardsdatascience, Vanishing gradient problem, 8 січня, 2019 https://towardsdatascience.com/the-vanishing-gradient-problem-69bf08b15484
7. Medium, Ahmet ÖZLÜ, 7 червня 2020 https://ahmetozlu93.medium.com/long-short-term-memory-lstm-networks-in-a-nutshell-363cd470ccac
8. Wikipedia, Root mean square deviation https://en.wikipedia.org/wiki/Root-mean-square_deviation
9. ARIMA model, Selva Prabhakaran https://www.machinelearningplus.com/time-series/arima-model-time-series-

forecasting-python/#:~:text=ARIMA%2C%20short%20for%20'Auto%20Regressive,used%20to%20forecast%20future%20values.

Додаток А
(довідниковий)
Зчитування даних з датасету

```python
# read data from csv file
def read_data():
    dataframe = read_csv('airline-passengers.csv', usecols=[1], engine='python')
    dataset = dataframe.values
    dataset = dataset.astype('float32')
    return dataset
```

Додаток Б
(довідниковий)
Масштабування даних

```python
# scale dataset to size 0 and 1
def scale_dataset(dataset):
    scaler = MinMaxScaler(feature_range=(0, 1))
    dataset = scaler.fit_transform(dataset)
    return dataset, scaler
```

Додаток В
(довідниковий)
Розділення даних на тренувальну та тестову частину

```python
def split_on_train_and_test(dataset):
    train_size = int(len(dataset) * 0.50)
    return dataset[0:train_size, :], dataset[train_size:len(dataset), :]
```

Додаток Г
(довідниковий)
Приведення даних до рекурентного вигляду

```
# convert an array of values into a dataset matrix
def create_dataset(dataset, look_back=1):
    dataX, dataY = [], []
    for i in range(len(dataset) - look_back - 1):
        a = dataset[i:(i + look_back), 0]
        dataX.append(a)
        dataY.append(dataset[i + look_back, 0])
    return numpy.array(dataX), numpy.array(dataY)
```

Додаток Ґ
(довідниковий)
Змінення форми датасету

```
# reshape input to be [samples, time steps, features]
def reshape_dataset(trainX, testX):
    trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
    testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
    return trainX, testX
```

Додаток Д
(довідниковий)
Створення та підготовка LSTM

```
# create and fit the LSTM network
def create_LSTM(trainX, trainY, look_back):
    model = Sequential()
    model.add(LSTM(4, input_shape=(1, look_back)))
    model.add(Dense(1))
    model.compile(loss='mean_squared_error', optimizer='adam')
    model.fit(trainX, trainY, epochs=50, batch_size=1, verbose=2)
    return model
```

Додаток Е
(довідниковий)
Створення прогнозування

```
# make predictions
trainPredict = model.predict(trainX)
testPredict = model.predict(testX)
```

Додаток Є
(довідниковий)
Зворотне масштабування

```
# invert predictions
trainPredict = scaler.inverse_transform(trainPredict)
trainY = scaler.inverse_transform([trainY])
testPredict = scaler.inverse_transform(testPredict)
testY = scaler.inverse_transform([testY])
```

Додаток Ж
(довідниковий)
Зміщення даних для відображення

```
# shift predictions for plotting
def shift_prediction(dataset, trainPredict, testPredict, look_back):
    # shift train predictions for plotting
    trainPredictPlot = numpy.empty_like(dataset)
    trainPredictPlot[:, :] = numpy.nan
    trainPredictPlot[look_back:len(trainPredict) + look_back, :] = trainPredict
    # shift test predictions for plotting
    testPredictPlot = numpy.empty_like(dataset)
    testPredictPlot[:, :] = numpy.nan
    testPredictPlot[len(trainPredict) + (look_back * 2) + 1:len(dataset) - 1, :] = testPredict
    return trainPredictPlot, testPredictPlot
```

Додаток З
(довідниковий)
Побудова графіка результатів

```
# build final plot
def build_plot(dataset, trainPredictPlot, testPredictPlot):
    plt.plot(scaler.inverse_transform(dataset), label='observed')
    plt.plot(trainPredictPlot, label='train', color="red")
    plt.plot(testPredictPlot, label='predicted')
    plt.xlabel("Month")
    plt.ylabel("Passengers")
    plt.legend()
    plt.show()
```

Додаток И
(довідниковий)
Вираховування середньоквадратичної похибки

```
# calculate root mean squared error
def calculate_RMSE(trainY, trainPredict, testY, testPredict):
    trainScore = math.sqrt(mean_squared_error(trainY[0], trainPredict[:, 0]))
    print('Train Score: %.2f RMSE' % (trainScore))
    testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:, 0]))
    print('Test Score: %.2f RMSE' % (testScore))
```

Додаток І
(довідниковий)
Зменшення похибки в прогнозуванні з епохами