МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ

КАФЕДРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
АВТОМАТИЗОВАНИХ СИСТЕМ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

тему Дослідження та моделювання зв'язків у соціумі за допомогою нейронних мереж

Виконав: студент 2 курсу, групи 8.1210-іпз
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
С. І. Руденко
(ініціали та прізвище)
Керівник доцент, В. І. Заяц
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент директор ТОВ «Дісітел»
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2021

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІМ. Ю.М. ПОТЕБНІ

Кафедра _____ програмного забезпечення автоматизованих систем
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність ___121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри_В.Г. Вербицький
"_01_" __вересня_____2021_року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

__Руденку Семену Ігоровичу__
(прізвище, ім'я, по батькові)

1.Тема роботи __Дослідження та моделювання зв'язків у соціумі за допомогою нейронних мереж_____

керівник роботи ___Заяц Валерій Іванович, доцент_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від ___"30" червня 2021 року № 974-с_____

2. Строк подання студентом кваліфікаційної роботи_____30.11.2020_____
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми побудови та навчання нейронної мережі та розробка методів її вирішення;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
___слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата Завдання прийняв

7. Дата видачі завдання 01.09.2021

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області	02.09-10.09.21	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	11.09-12.09.2021	виконано
3	Аналіз існуючих методів рішення	13.09-17.09.20	виконано
4	Дослідження області нейронної мережі	18.09-24.09.20	виконано
5	Узгодження подальших дій з науковим керівником	25.09-26.09.20	виконано
6	Аналіз теоретичних відомостей	27.09-15.10.20	виконано
7	Проектування інтерфейсу додатку з навченою нейронною мережею	15.10-23.10.20	виконано
8	Узгодження інтерфейсу з науковим керівником	23.10-24.10.20	виконано
9	Реалізація функціоналу побудови нейронної мережі	25.10-14.11.20	виконано
10	Представлення отриманих результатів науковому керівнику і узгодження плану подальшого дослідження	15.11-16.11.20	виконано
11	Реалізація функціоналу навчання нейронної мережі	16.11-23.11.20	виконано
12	Проведення аналізу можливостей розроблених програмних застосунків	24.11-09.12.21	виконано
13	Оформлення звіту	10.12-28.12.21	виконано

Студент _____ С.І. Руденко
(підпис) (прізвище та ініціали)

Керівник роботи _____ В.І. Заяц
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ І.А. Скрипник
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 75

Рисунків: 32

Джерел: 18

Руденко С. І. Дослідження та моделювання зав’язків у соціумі за допомогою нейронних мереж : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник В. І. Заяц. Запоріжжя : ЗНУ. 2021. 75 с.

Метою даної роботи є дослідження та моделювання зав’язків у соціумі за допомогою нейронних мереж. Об’єктом дослідження є нейронна мережа створена для керування поведінкою соціальними групами. Для отримання необхідних результатів дослідження використовувалися наступні методи: аналіз нейронних мереж, моделювання нейронної мережі, методи навчання нейронних мереж. Створена нейронна мережа для керування соціальними групами та вирішення проблем для кожної соціальної групи. В результаті розроблено додаток, котрий представляє собою застосунок створений за допомогою фреймворку Flutter, який демонструє процес навчання нейронної мережі та демонструє кінцевий результат навчання користувачеві.

Ключові слова: *нейронна мережа, нейрон, соціум, навчання, Q-learning.*

SUMMARY

Pages: 75

Figures: 32

Sources: 18

Rudenko S.I. Research and modeling of connections in society with the help of neural networks: qualification work of the master of specialty 121 "Software Engineering" / science. head V.I. Zayats. Zaporizhzhia : ZNU. 2021. 75 p.

The aim of this work is to study and model connections in society using neural networks. The object of research is a neural network created to control the behavior of social groups. The following methods were used to obtain the necessary results of the study: neural network analysis, neural network modeling, neural network training methods. A neural network has been created to manage social groups and solve problems for each social group. As a result, an application was developed, which is an application created using the Flutter framework, which demonstrates the learning process of the neural network and demonstrates the end result of learning to the user.

Keywords: *neural network, neuron, society, learning, Q-learning.*

ЗМІСТ

ВСТУП

Актуальність теми

Нейронні мережі ідеально підходять для вирішення складних завдань в реальних життєвих ситуаціях. Вони можуть вивчати і моделювати нелінійні і складні залежностями між вхідними і вихідними данним; робити узагальнення і висновки; виявити приховані зв'язки, закономірності і прогнози; і моделювати дуже мінливі дані (наприклад, дані фінансових часових рядів) і відхилення, необхідні для прогнозування рідкісних подій (наприклад, виявлення шахрайства).

Сьогодні нейронні мережі застосовуються в наступних сферах:

- виявлення шахрайства з кредитними картами;
- оптимізація логістики транспортних мереж;
- розпізнавання символів і голоси, також відоме як обробка природної мови;
- медична діагностика та діагностика захворювань;
- цільовий маркетинг;
- фінансові прогнози цін на акції, валюти, опціонів, ф'ючерсів, банкрутства та рейтинги облігацій;
- роботизовані системи управління;
- прогнозування електричного навантаження і попиту на електроенергію;
- контроль процесів і якості;
- ідентифікація хімічної сполуки;
- оцінка екосистеми;
- комп'ютерне зір для інтерпретації необроблених фотографій і відео (наприклад, у медичній візуалізації, робототехніці і розпізнаванні осіб).

Так як сучасні нейронні мережі мають дуже великі можливості і різні варіанти використання, їх популярність зростає, а розвиток галузі теж йде високим темпом. Їх вчать грати в комп'ютерні ігри, розпізнавати голоси і т. Д. По суті, штучні мережі створюються за принципом біологічних, а значить, ми можемо навчити їх виконувати ті процесі, які людина виконує не завжди може реалізувати.

Мета і завдання дослідження

Метою даної роботи є дослідження та моделювання зв’язків у соціумі за допомогою нейронних мереж.

Об’єкт дослідження

Об’єктом дослідження є нейронна мережа створена для керування поведінкою соціальними групами.

Предмет дослідження

Предметом дослідження є нейронні мережі.

Методи дослідження

Для отримання необхідних результатів дослідження використовувалися наступні методи:

- Аналіз нейронних мереж
- Дослідження існуючих нейронних мереж
- Моделювання нейронної мережі
- Пошук та аналіз наукової літератури

Наукова новизна одержаних результатів

Створення нейронної мережі для керування соціальними групами та вирішення проблем для кожної соціальної групи.

Практичне значення одержаних результатів

В результаті будемо розроблено додаток, котрий представляє собою застосунок створений за допомогою фреймворку Flutter, який демонструє процес навчання нейронної мережі та демонструє кінцевий результат навчання користувачеві.

Апробація одержаних результатів

Підсумки дослідження були представлені на XIV університетській науково-практичній конференції студентів, аспірантів, докторантів і молодих вчених Запорізького національного університету «Молода наука-2021» [1], а також на І Всеукраїнській науково-практичній конференції здобувачів вищої освіти, аспірантів та молодих вчених Інженерного навчально-наукового інституту Запорізького національного університету [2].

Глосарій

Нейронна мережа — кінцевий автомат, наступний стан у якого в загальному випадку невизначено і, відповідно, не визначений вихідний сигнал.

Нейрон — біологічний нейрон імітується у штучній нейронній мережі через активаційну функцію. У задачах класифікації (наприклад визначення спам-повідомлень) активаційна функція повинна мати характеристику "вмикача". Іншими словами, якщо вхід більше, ніж деяке значення, то вихід повинен змінювати стан, наприклад з 0 на 1 або -1 на 1. Це імітує "включення" біологічного нейрону. У якості активаційної функції зазвичай використовують сигмоїдну функцію.

Вхідний шар — шар нейронів (вузлів), утворений сукупністю елементів, на входи яких подаються вхідні дані нейронної мережі.

Прихований (проміжний) шар — шар, що містить нейрони (вузли), на які не надходять вхідні дані і з яких не зчитуються вихідні дані нейронної мережі.

Вихідний шар – шар нейронів (вузлів), вихідні сигнали яких утворюють вихідний сигнал нейронної мережі.

Персептрон — найпростіша форма нейронної мережі. В основному під персептроном розуміють елементарний нейрон, що представляє собою лінійний суматор, кожен з вхідних сигналів якого множиться на певний ваговій множник, а вихідний сумарний сигнал є ненульовим, якщо сума перевищує деяке порогове значення. Іноді персептроном називають будь-яку НС шару-

ватої структури. Однак тут і далі під персептроном розуміється тільки одношарова (single-layer perceptron) або багатошарова (multilayer perceptron) мережа, що складається з нейронів з активаційними функціями одиничного стрибка (бінарна мережу).

Мережа з прямими зв'язками (мережа без зворотних зв'язків) — багатошарова мережа, в якій кожен шар своїми входами має виходи тільки попередніх шарів.

Мережа зі зворотним зв'язком — мережа з сполуками з виходу мережі на її вхід.

Рекурентна нейронна мережа — нейронна мережа, що відрізняється від мережі з прямими зв'язками наявністю хоча б однієї петлі зворотного зв'язку.

Навчання (тренування) — етап функціонування нейронної мережі, в процесі якого на її вхід по черзі надходять дані з навчального набору з метою коригування вагових коефіцієнтів синаптичних зв'язків для отримання найбільш адекватного сигналу на виході нейронної мережі.

Навчання з вчителем — процес навчання нейронної мережі, є обов'язковою вимогою якого є існування готового навчального набору даних.

Навчання без учителя — процес навчання нейронної мережі, при якому наявність набору еталонів відсутня.

Перенавчання — ситуація, коли на навчальної послідовності помилки мережі були дуже малі, але на нових даних стають великими.

Git — це безкоштовна розподілена система контролю версій з відкритим вихідним кодом, призначена для швидкої та ефективної роботи з усім, від малих до дуже великих проектів.

GitHub — найбільший веб-сервіс для хостингу IT-проектів та їх спільної розробки. Веб-сервіс, заснований на системі контролю версії Git і розроблений на Ruby on Rails та Erlang компанією GitHub, Inc.

Dart — мова програмування загального призначення від компанії Google, яка призначена перш за все для розробки веб-додатків (як на стороні клієнта, так і на стороні сервера) та мобільних додатків. Dart позиціонується як заміна/альтернативу JavaScript.

Flutter — комплект засобів розробки та фреймворк з відкритим вихідним кодом для створення мобільних додатків під Android та iOS, веб-додатків та нативних додатків під Windows, Apple та Linux з використанням мови програмування Dart, розроблений і розвивається корпорацією Google. Основні складові комплекту – платформа Dart, двигун Flutter, бібліотека Foundation, набори віджетів та засоби розробки (Flutter DevTools).

`Bloc` – бібліотека, мета якої полягає в тому, щоб спростити поділ презентації від бізнес-логіки, полегшуючи тестування та можливість повторного використання.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ НЕЙРОННОЇ МЕРЕЖІ

1.1 Що таке нейронна мережа

Штучна нейронна мережа — це програмне втілення математичної моделі, яка побудована за принципом нейронних мереж з біології. Подібна структура надає можливість машині запам’ятовувати, аналізувати та відтворювати з пам’яті інформацію подібно людському мозку, у якому нейрони передають інформацію у вигляді електричних імпульсів.

Нейронна мережа навчається знаходити зв’язки між нейронами — це відрізняє її від звичних алгоритмів, що діють за певною схемою. Під час навчання мережа спроможна знаходити складні залежності та спираючись на них знаходити правильний результат навіть при відсутності або невеликій кількості даних.

1.2 Принцип роботи нейронної мережі

Нейромережа — це сукупність нейронів, що отримують дані, обробляють їх, а потім передає до іншого нейрона. Нейрони з’єднані між собою за допомогою синапсів. Кожен синапс може послаблюють або підсилюють сигнал нейрону.

Нейронна мережа складається з трьох компонентів, представлених на рисунку 1:

- вхідний шар;
- приховані шари;
- вихідний шар.

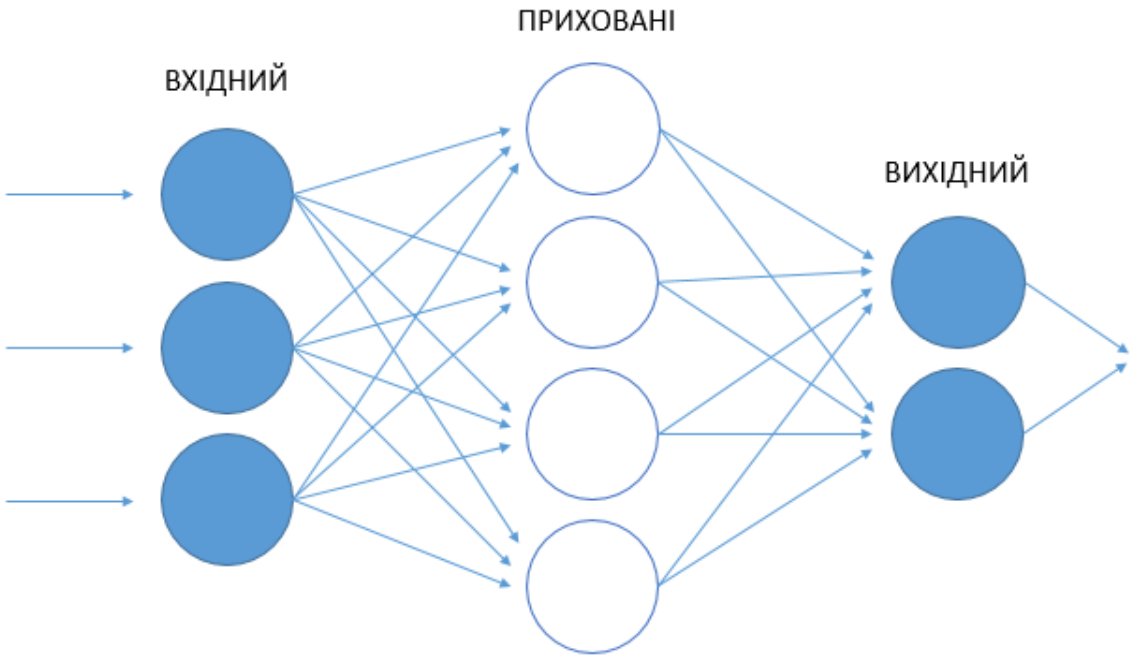

Рис.*1 Структура нейронної мережі*

На вхідних шарах нейронів відбувається надходження певних даних. Інформація передається далі за допомогою синапсів наступного шару, також кожен синапс має власний коефіцієнт ваги, а будь-який наступний нейрон здатний мати кілька вхідних синапсів. Дані, отримані в таким нейроном, — це сума всіх даних для нейронних мереж, які перемножити на коефіцієнти ваг. Отримане в результаті значення передається до функції активації, в результаті чого відбувається створення вихідної інформації. Далі інформація передається до тих пір, поки не дійде до виходу.

Перший запуск нейронної мережі не дає вірних результатів, адже вона ще не навчилась. Якщо казати про поняття функції активації, то ця функція використовується для нормалізації вхідних даних. Таких функцій буває дуже багато, розглянемо найбільш поширені. Головна відмінність цих функцій це область їх визначення:

- лінійна функція f (x) = x. Є найбільш простою з усіх, вона повинна застосовуватися лише для тестування створеної нейронної мережі або передачі даних у вихідній формі;
- сигмоід — більш розповсюджена функція активації. Діапазон значень для якої від нуля до одиниці. Інколи її називаю логістичною функцією;
- гіперболічний тангенс. Ця функція необхідна для охоплення, окрім позитивних, також і негативних значень. Коли їх застосування не передбачено при розробці, гіперболічний тангенс не потрібен.

1.3 Опис проблеми

Щоб комп’ютер міг вирішити якусь проблему йому потрібно показати що потрібно робити, щоб досягти результату. Основною його проблемою є те що він не вміє думати та лише виконує команди, які йому надсилає користувач. Чи може комп’ютер мислити? Для вирішення цього питання люди створюють штучний інтелект(ШІ). Щоб створити його використовується нейронна мережа. Шляхом аналізу мозку людини вчені прийшли до висновку що наш мозок складається з нейронів. По принципу людського мозку створюється нейронна мережа. Вона аналізує вхідні дані через систему нейронів. Після аналізу, нейронна мережа повертає результат обчислень на основі взаємодії нейронів та досвіду й помилок минулих запусків системи [3]. Це означає, що будь-яка нейронна мережа є системою, що самостійно навчається. Нейронні мережі використовують майже у всіх галузях для вирішення нетипових задач.

Рішення нейронних мереж важко інтерпретувати ретроспективно. Тому методи глибокого навчання часто піддаються критиці за те, що вони є «чорним ящиком». На відміну від деяких більш простих алгоритмів навчання, зо-

крема дерев рішень, вони не пропонують простого і зрозумілого пояснення; їх архітектура складна і складається з декількох або багатьох рівнів з сотнями тисяч параметрів, які необхідно навчити.

1.4 Приклади використання нейронних мереж

Stockfish. Stockfish — це найпотужніший шаховий рушій з відкритим вихідним кодом. Stockfish постійно активно розвивається та працює на багатьох платформах. Його інтерфейс зображено на рисунку 2 [16].

Рис.*2 Stockfish*

Stockfish розроблений Марко Костальбою, Джуно Кійскі, Гарі Ліндскоттом і Тордом Ромстадом, але велики внесок зробили і інші розробники. До 2017 року вважався найсильнішим шаховим рушієм та вигравав безліч чемпіонатів доки не програв нейронній мережі під назвою AlphaZero. Однак Stockfish все ще займає значні місця.

Stockfish — це шахматний рушій з підтримкою UCI. UCI — це комунікаційний протокол, що дозволяє шахматним рушіям взаємодіяти з графічним інтерфейсом.

Гра Quick, Draw. Гра Quick, Draw!, зображена на рисунку 3, була створена компанією Google. Вся суть гри полягає в малюванні об’єкту, запропонованого нейронною мережею, за обмежену кількість часу [17].

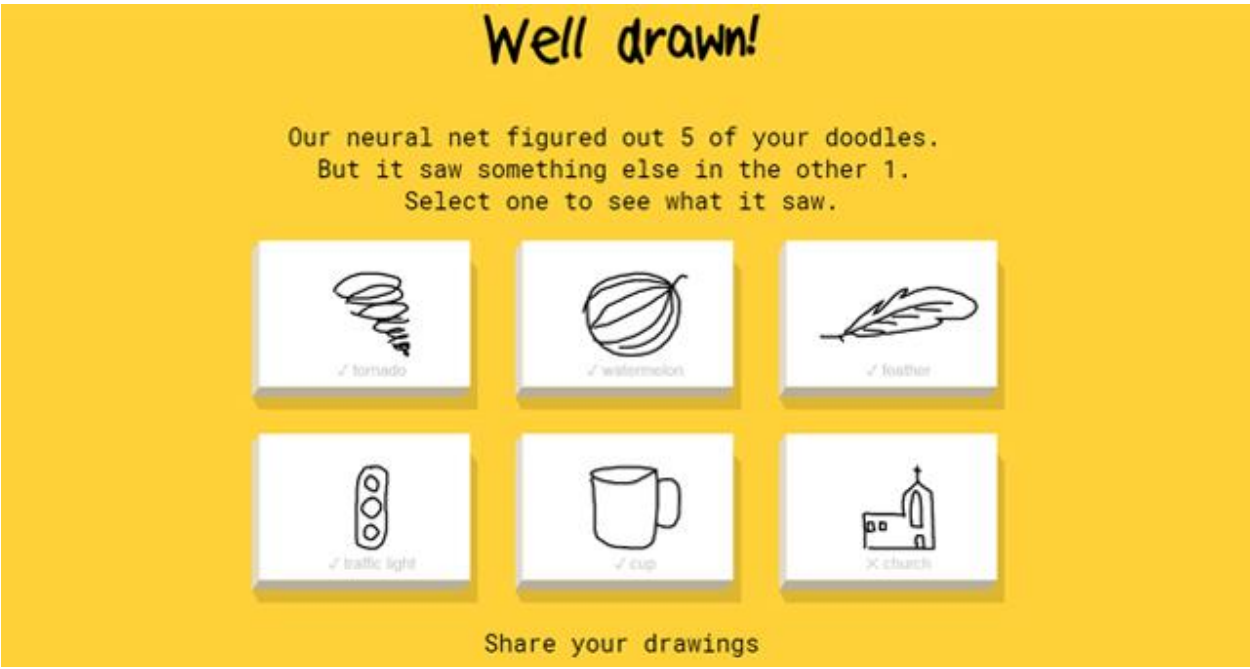

Рис.*3 Quick, Draw!*

У грі використовується нейронна мережа, що лягла за основу у перекладачі від Google, а саме розпізнаванні рукописного тексту.

За двадцять секунд користувач повинен намалювати предмет, що обирає для нього гра у випадковому порядкую. Нейронна мережа також враховує не тільки те, що гравець малює, але й послідовність його штрихів.

Розпізнавання малюнка — непроста задача для нейромережі, адже будь-який предмет можна зобразити багатьма різними способами, які швидко і точно буде впізнавати людина. Для цього нейронній мережі постійно треба навчатися, чому гра і сприяє.

Після завершення гри малюнки користувача додаються до бази даних для детального аналізу кожного з них та вподальшему удосконалення алгоритмів нейронної мережі.

IBM Watson. Кінокомпанія 20th Century Fox за допомогою суперкомп'ютера IBM Watson створила трейлер до фантастичного трилеру «Морган», що зображен на рисунку 4. Штучний інтелект переглянув весь фільм та запропонував 10 фрагментів відео з фільму, що найкраще підходять. Після чого фахівці IBM змонтували ці фрагменти та отримали повноцінний трейлер.

Рис.4 *Кадри з фільму «Морган»*

Shazam. Shazam — це найпопулярніший додаток для пошуку музики, що працює на базі нейронних мереж. Принцип дії — це створити спектрограму пісні, очистивши її від сторонніх шумів, а потім порівняти її з мільйоном інших пісень застосовуючи певні алгоритми. Цей спосіб підходить для пошуку пісень, що лунають на вулиці або по радіо. Інтерфейс додатку надано на рисунку 5 [18].

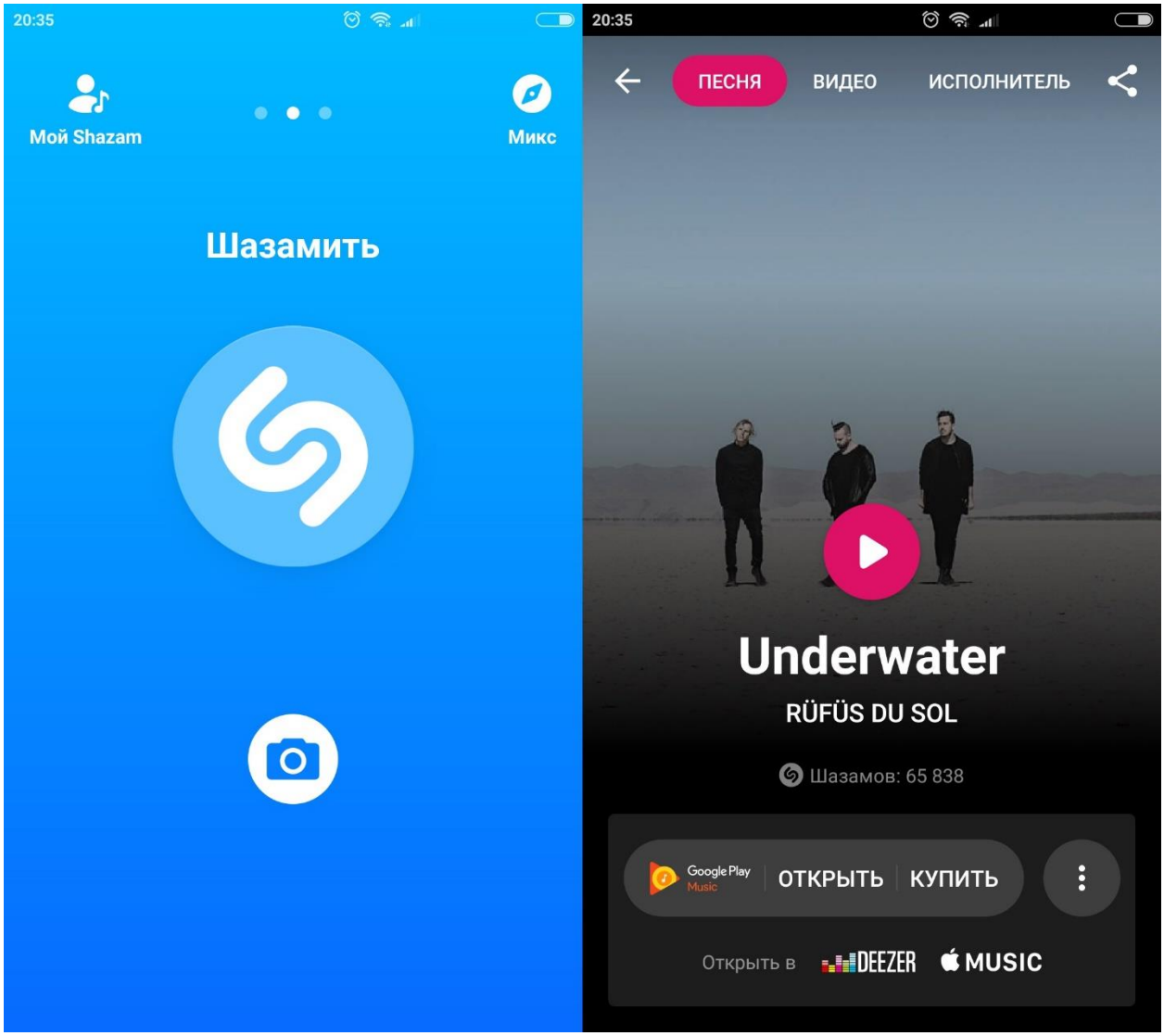

Рис.*5 Shazam*

Звук — це коливальний рух частинок середовища, що поширюється у вигляді хвилі. Надходячи до барабанної перетинки у наших вухах, ці коливання рухають маленькі кісточки, що передають вібрацію до волоскових клітин у нашому внутрішньому вусі. Волоскові клітини виробляють електричні імпульси, що передаються до нашого мозку за допомогою слухового нерву.

Записуючі пристрої досить точно імітують цей процес, використовуючи тиск звукової хвилі для перетворення її в електричний сигнал. Фактична звукова хвиля в повітрі є безперервним сигналом тиску. У мікрофоні перший електричний компонент, який зустрічає цей сигнал, перетворює його в аналоговий сигнал напруги – знову ж таки, безперервний. Цей безперервний сиг-

нал не настільки корисний у цифровому світі, тому перед його обробкою його потрібно перевести в дискретний сигнал, який можна зберегти в цифровому вигляді. Це робиться шляхом захоплення цифрового значення, яке представляє амплітуду сигналу.

Перетворення включає в себе квантування вхідних даних, і воно обов'язково вносить невелику кількість помилок. Таким чином, замість одного перетворення, аналого-цифровий перетворювач виконує безліч перетворень на дуже малих частинах сигналу – процес, відомий як вибірка.

This Person Does Not Exist / This Cat Does Not Exist. Сайти, що створюють зображення неіснуючих людей та котів завдяки нейромережі. Нейронна мережа StyleGAN від Nvidia обробляє мільйони зображень, а потім виводить зображення вигаданих людей або котів. На рисунку 6 зображено результат роботи сайту This Person Does Not Exist, на рисунку 7 зображено відображення назви нейронної мережі, що створила людину на рисунку 6.

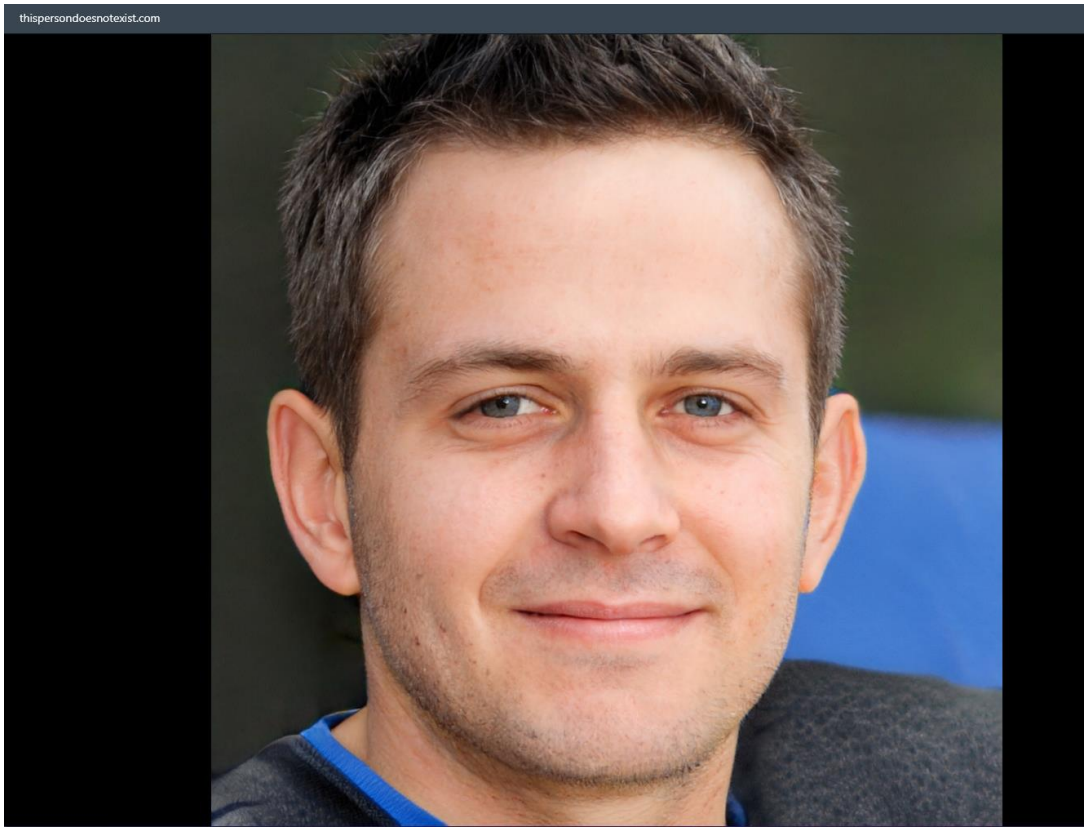

Рис.*6 This Person Does Not Exist*

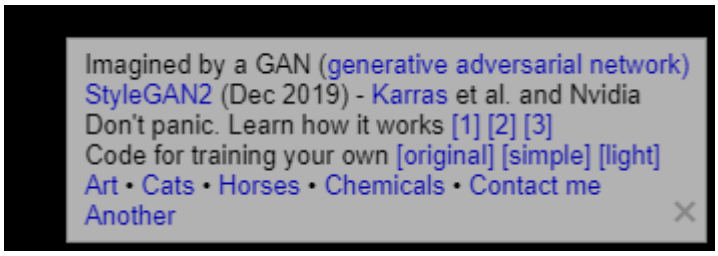

Рис.*7 Назва нейронної мережі, яка створила людину*

Генерування оптимальних конфігурацій стрижнів за допомогою штучного інтелекту

Зараз ядерна енергія дає більше безвуглецевої електроенергії в США, ніж сонячна та вітрова разом узяті. Це робить її ключовим гравцем у боротьбі зі зміною клімату. Проте методи її видобутку недосконалі та старіють. Необхідно оптимізувати процес, щоб ядерна енергетика могла конкурувати з вугільними та газовими електростанціями на ринку.

Скоротити витрати на видобуток можна оптимізувавши паливні стрижні на глибині ядерного реактора. Вони запускають реакції і коли розташовані ідеально, спалюють менше палива і вимагають менше обслуговування. Через десятиліття проб і помилок інженери-ядерники навчилися розробляти досконаліші схеми розташування дорогих паливних стрижнів, щоб продовжити термін служби. Тепер їм допоможе штучний інтелект (ІІ).

Дослідники з Массачусетського технологічного інституту (MIT) і Exelon впевнені — перетворивши процес проектування на гру, систему штучного інтелекту можна навчити генерувати десятки оптимальних конфігурацій стрижнів, які можуть продовжити термін служби кожного з них приблизно на 5%. Це дозволяє заощадити на типовій електростанції близько $3 млн. на рік. Система штучного інтелекту може знаходити оптимальні рішення швидше, ніж людина, і швидко змінювати конструкції в безпечному змодельованому середовищі.

У типовому реакторі паливні стрижні вибудовані в сітку або складання за рівнями урану та оксиду гадолінію всередині, як шахові фігури на дошці, з реакціями, що запускають радіоактивний уран, та рідкоземельним гадолінієм, що уповільнює їх. В ідеальному компонуванні ці конкуруючі імпульси врівноважуються, щоб стимулювати ефективні реакції. Інженери намагалися використовувати традиційні алгоритми для поліпшення макетів, розроблених людиною, але у стандартній збірці зі 100 стрижнів може бути астрономічна кількість варіантів для оцінки.

Дослідники поцікавилися, чи може глибоке навчання з підкріпленням — техніка штучного інтелекту, яка дозволила досягти надлюдської майстерності в таких іграх, як шахи і го, — прискорити процес перевірки. Глибоке навчання з підкріпленням поєднує у собі глибокі нейронні мережі, які чудово виділяють закономірності в масивах даних, з навчанням із підкріпленням, яке пов'язує навчання із сигналом винагороди, таким як перемога у грі.

У новому експерименті дослідники навчили свого агента розміщувати паливні стрижні відповідно до набору обмежень, заробляючи більше очок за кожен вдалий перебіг. Кожне обмеження чи правило, обране дослідниками, відбиває десятиліття експертних знань, заснованих на законах фізики. Агент може набирати бали, наприклад, розміщуючи стрижні з низьким вмістом урану на краях складання, щоб уповільнити реакції.

Завдяки навчанню з підкріпленням штучний інтелект навчився грати у все більш складні ігри не гірше за людей або навіть краще. Але його можливості залишаються марними у реальному світі. Тепер дослідники довели, що навчання з підкріпленням має потенціал.

Exelon зараз тестує бета-версію системи штучного інтелекту у віртуальному середовищі. За словами представника компанії, система може бути готова до запровадження через рік чи два.

1.5 Типи нейронних мереж

За своєю структурою нейронні мережі поділяються на:

- Одношарові нейронні мережі. Являє собою структуру взаємодії нейронів, в якій сигнали зі вхідного шару відразу направляються на вихідний шар, який не тільки перетворює сигнал, але і відразу ж видає відповідь. Перший вхідний шар тільки приймає і розподіляє сигнали, а потрібні обчислення відбуваються вже в другому шарі. Вхідні нейрони є об'єднаними з основним шаром за допомогою синапсів з різними вагами, що забезпечують якість зв’язків.
- Багатошарові нейронні мережі. В ній крім вихідного і вхідного шарів, є ще кілька прихованих проміжних шарів. Число цих шарів залежить від ступеня складності нейронної мережі. Вона більшою мірою нагадує структуру біологічної нейронної мережі. Такі види були розроблені зовсім недавно, до цього всі процеси були реалізовані за допомогою одношарових нейронних мереж. Відповідні рішення мають більші можливості, якщо порівнювати з одношаровими, адже в процесі обробки даних кожен проміжний шар — це проміжний етап, на якому здійснюється обробка і розподіл інформації.

По напрямку розподілу інформації по синапсах між нейронами:

- Нейронні мережі прямого поширення (односпрямовані). Сигнал переміщається строго по напрямку від вхідного шару до вихідного. Рух сигналу в зворотному напрямку не здійснюється і в принципі неможливо. Сьогодні розробки цього плану поширені широко і на сьогоднішній день успішно вирішують завдання розпізнавання образів, прогнозування та кластеризації.
- Рекурентні нейронні мережі (із зворотними зв'язками). Сигнал рухається і в прямому, і в зворотному напрямку. У підсумку ре-

зультат виходу здатний повертатися на вхід. Вихід нейрона визначається ваговими характеристиками і вхідними сигналами, плюс доповнюється попередніми виходами, знову повернулися на вхід. Цій нейронній мережі властива функція короткочасної пам'яті, на підставі чого сигнали відновлюються і доповнюються під час їх обробки.

- Радіально-базисні функції.
- Самоорганізуючі карти.

Залежно від типів нейронів:

- однорідні;
- гібридні.

Залежно від методу нейронних мереж з навчання:

- навчання з учителем;
- без вчителя;
- з підкріпленням.

За типом вхідної інформації нейронні мережі бувають:

- аналогові;
- виконавчі;
- образні.

За характером налаштування синапсів:

- з фіксованими зв'язками;
- з динамічними зв'язками.

1.6 Навчання із частковим залученням вчителя

Навчання з частковим залученням вчителя (semi-supervised learning) характеризується своєю назвою: навчальний датасет містить як розмічені, і нерозмічені дані. Цей метод особливо корисний, коли важко отримати з даних важливі ознаки або розмітити всі об'єкти — трудомістке завдання.

Навчання з частковим залученням вчителя часто використовують для вирішення медичних завдань, де невелика кількість розмічених даних може призвести до значного підвищення точності.

Цей метод машинного навчання є поширеним для аналізу медичних зображень, таких як скани комп'ютерної томографії або МРТ. Досвідчений рентгенолог може розмітити невелике підмножина сканів, на яких виявлено пухлини та захворювання. Але вручну розмічати всі скани — надто трудомістке та дороге завдання. Тим не менш, нейронна мережа може витягти інформацію з невеликої частки розмічених даних і поліпшити точність передбачень у порівнянні з моделлю, що навчається виключно на нерозмічених даних.

Популярний метод навчання, для якого потрібен невеликий набір розмічених даних, полягає у використанні генеративно-змагальної мережі або GAN.

Уявімо собі змагання двох нейронних мереж, де кожна намагається перехитрити іншу. Це GAN. Одна з мереж, генератор, намагається створити нові об'єкти даних, які імітують навчальну вибірку.

Інша мережа, дискримінатор, оцінює, чи ці згенеровані дані є реальними або підробленими. Мережі взаємодіють і циклічно вдосконалюються, оскільки дискримінатор намагається краще відокремлювати підробки від оригіналів, а генератор намагається створювати переконливі підробки.

Принцип роботи GAN зображено на рисунку 8. Дискримінатору "D" показують вихідні зображення та дані, створені генератором "G". Дискримінатор повинен визначити, які зображення є реальними, а які є підробленими.

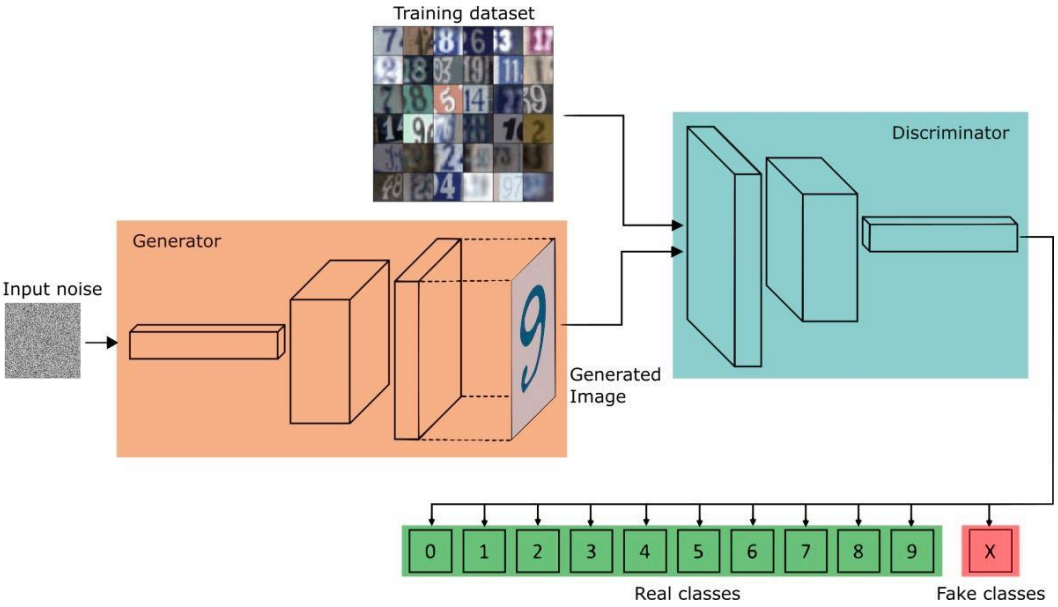

Рис.*8 GAN*

1.7 Переваги нейронних мереж

Рішення задач при невідомих закономірностях. Використовуючи здатність навчання на безлічі прикладів, нейронна мережа здатна вирішувати завдання, в яких невідомі закономірності розвитку ситуації і залежності між вхідними та вихідними даними.

Стійкість до шумів у вхідних даних. Можливість роботи при наявності великого числа неінформативних, шумових вхідних сигналів. Немає необхідності робити їх попередній відсів, нейронна мережа сама визначить їх малопридатними для вирішення завдання і відкине їх [15].

Адаптація до змін навколишнього середовища. Нейронні мережі мають здатність адаптуватися до змін навколишнього середовища. Зокрема, нейронні мережі, навчені діяти в певному середовищі, можуть бути легко перевивчені для роботи в умовах незначних коливань параметрів середовища. Більш того, для роботи в нестаціонарному середовищі (де статистика змінюється з плином часу) можуть бути створені нейронні мережі, переучувати в реальному часі. Чим вище адаптивні здібності системи, тим більш стійкою буде її робота в нестаціонарному середовищі. При цьому слід зауважити, що

адаптивність не завжди веде до стійкості; іноді вона призводить до абсолютно протилежного результату. Наприклад, адаптивна система з параметрами, швидко змінюються в часі, може також швидко реагувати і на сторонні збудження, що викличе втрату продуктивності. Для того щоб використовувати всі переваги адаптивності, основні параметри системи повинні бути досить стабільними, щоб можна було не враховувати зовнішні перешкоди, і досить гнучкими, щоб забезпечити реакцію на істотні зміни середовища.

Відмовостійкість при апаратній реалізації нейронної мережі. Нейронні мережі потенційно відмовостійкі. Це означає, що при несприятливих умовах їх продуктивність падає незначно. Наприклад, якщо пошкоджений якийсь нейрон або його зв'язку, витяг збереженої інформації ускладняється. Однак, беручи до уваги розподілений характер зберігання інформації в нейронної мережі, можна стверджувати, що тільки серйозні пошкодження структури нейронної мережі істотно вплинуть на її працездатність. Тому зниження якості роботи нейронної мережі відбувається повільно.

1.8 Постановка задачі

Дослідити принципи роботи нейронної мережі, розробити додаток з нейронною мережею для керування соціальними групами та вирішення проблем для кожної з них.

РОЗДІЛ 2 АНАЛІЗ ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ НЕЙРОННОЇ МЕРЕЖІ

2.1 Основні поняття

Навчання нейронних мереж. Навчання нейронної мережі — це пошук найкращого набору ваг для максимізації точності передбачення. Нейронні мережі можуть бути використані і без чіткого розуміння, як саме вони навчаються.

Навчання нейромереж відбувається в два етапи:

- пряме поширення помилки;
- зворотне поширення помилки.

Під час прямого поширення помилки робиться передбачення відповіді. При зворотному поширенні помилка між фактичною відповіддю і передбаченим мінімізується.

Парадигми навчання нейронних мереж. Відомо три способи навчання нейронних мереж, в основу яких закладені особливості машинного навчання:

- з вчителем (supervised learning);
- без вчителя (unsupervised);
- з підкріпленням (reinforcement leaning).

Навчання з вчителем. Навчання з вчителем (supervised learning) — це для кожного вхідного вектора існує вектор вихідних значень. Разом ці два вектора називають парою навчання, а множину навчальних пар — навчальною вибіркою. Процес навчання зводиться до почергового подавання на вхід нейронної мережі навчальних пар, вираховування похибки між дійсним і бажаним значенням нейронної $\delta=y-d$ та корегування параметрів мережі в бік зменшення цієї похибки.

Навчання без вчителя. Навчання без вчителя (unsupervised) — один зі способів машинного навчання, при вирішенні яких випробовувана система спонтанно навчається виконувати поставлене завдання, без втручання з боку експериментатора. Дивлячись з точки зору кібернетики, є одним з видів кібернетичного експерименту. Це підходить тільки для задач, в яких відомий опис множини об'єктів, і необхідно виявити внутрішні взаємозв'язки, закономірності, що існують між різними об'єктами.

Навчання без вчителя не рідко протиставляється навчанню з вчителем, коли для кожного об'єкта, що навчається, примусово задається «правильна відповідь», та потрібно знайти залежність між стимулами та реакціями системи [2].

Навчання з підкріпленням. Навчання з підкріпленням є проміжним варіантом двох попередніх парадигм . Замість «вчителя» в схему навчання вводиться блок «критика» , який відслідковує реакцію середовища на вхідний сигнал і опираючись на неї визначає евристичну похибку, яку покладено в процес навчання мережі. Вказані парадигми базуються на відповідних правилах навчання, які визначають основні особливості їх застосування.

Швидкість навчання (learning rate). Інженер навчає нейронну мережу, надаючи навчальні дані і виконуючи процедуру навчання. У той час як це відбувається, мережа вчиться — або, більш конкретно, вона вчиться апроксимувати зв'язок вхід-вихід, що міститься в навчальних даних. Проявом навчання є зміна ваги, а швидкість навчання впливає на спосіб цієї зміни. Нормальна швидкість навчання зображена на рисунку 9.

Якщо швидкість навчання досить висока, то мережа дуже швидко видає відповіді. Отриманий результат зображено на рисунку 10.

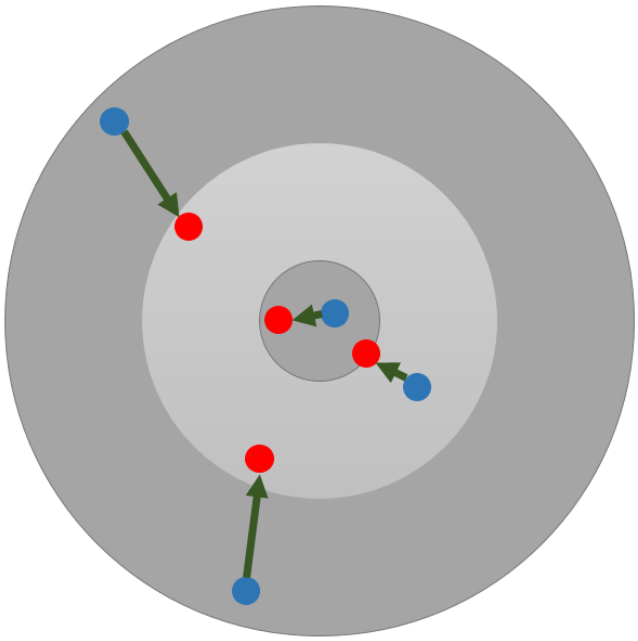

Рис. *9 Швидкість навчання*

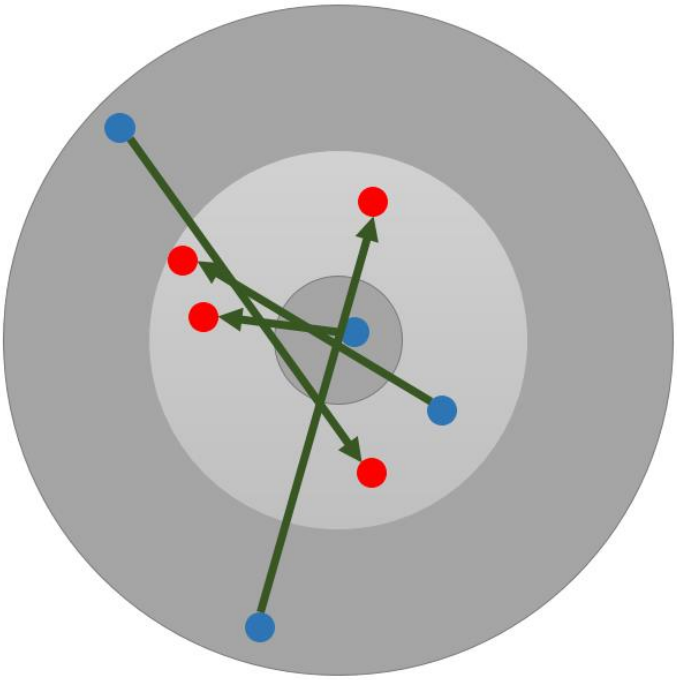

Рис. *10 Занадто висока швидкість навчання*

2.2Навчання з підкріпленням

Агент навчання з підкріпленням - це процес оптимізації, який навчається на досвіді, використовуючи дані зі свого середовища, які він зібрав за допомогою власних спостережень. Він починає з того, що нічого не знає про завдання явно, дізнається методом спроб і помилок про те, що відбувається, коли він приймає рішення, відстежує успішні рішення і приймає ті ж самі рішення за тих самих обставин у майбутньому.

Ключові поняття

Агент — це суб'єкт, який приймає рішення.

Навколишнє середовище — це світ, в якому діє агент, наприклад, гра для перемоги або завдання, яке потрібно виконати.

Стан — це місце, де агент знаходиться у своєму середовищі. Коли ви визначаєте стани, в яких агент може бути, подумайте про те, що йому потрібно знати про своє середовище. Наприклад, самоврядний автомобіль повинен знати, чи є наступний світлофор червоним або зеленим і чи є пішоходи на пішохідному переході; вони визначаються як змінні стани.

Дія — це наступний крок, який агент вибирає.

Винагорода — це зворотний зв'язок, яку агент отримує від середовища за цю дію.

Стратегія — це функція для зіставлення станів агента з його процесами. Для першого агента RL це буде так само просто, як таблиця підстановки, звана Q-таблицею. Він буде працювати як мозок вашого агента.

Цінність — це майбутня нагорода, яку агент отримає, здійснюючи дію, засновану на майбутніх діях, які він може зробити. Це окремо від негайної винагороди, яку він отримає від ухвалення цієї дії.

Перший тип агента навчання з підкріпленням — це агент без моделі. Агент без моделі нічого не знає про стан, який він не бачив і тому не зможе оцінити вартість винагороди, яку він отримає від невідомого стану.

Два основні алгоритми навчання з підкріпленням без моделі називаються Q-learning та state-action-reward-state-action (SARSA) [8].

Коли моделюється функція дії стану для будь-якої системи, вибираються змінні, які ми хочемо відслідковувати, і це дозволяє нам визначити, у яких станах може знаходитися система.

Функція дії стану агента називається його стратегією.

Стратегія може бути простою і прямолінійною, або складною і важкою для перерахування, залежно від самої проблеми і кількості станів і дій.

Високорівневий алгоритм навчального агента виглядає так:

1. Зверніть увагу, у якому стані ви знаходитесь.
2. Здійсніть дію, засновану на вашій стратегії, та отримайте винагороду.
3. Зверніть увагу на винагороду, яку ви отримали, здійснивши цю дію у цьому стані.

Це можна виразити математично, використовуючи процес ухвалення рішень Маркова.

2.3 Опис сервісів для роботи з нейронними мережами

TensorFlow. Для реалізації нейронної мережі існують багато готових рішень. Один най популярніших та безкоштовних сервісів є TensorFlow . Інтерфейс зображено на малюнку 11.

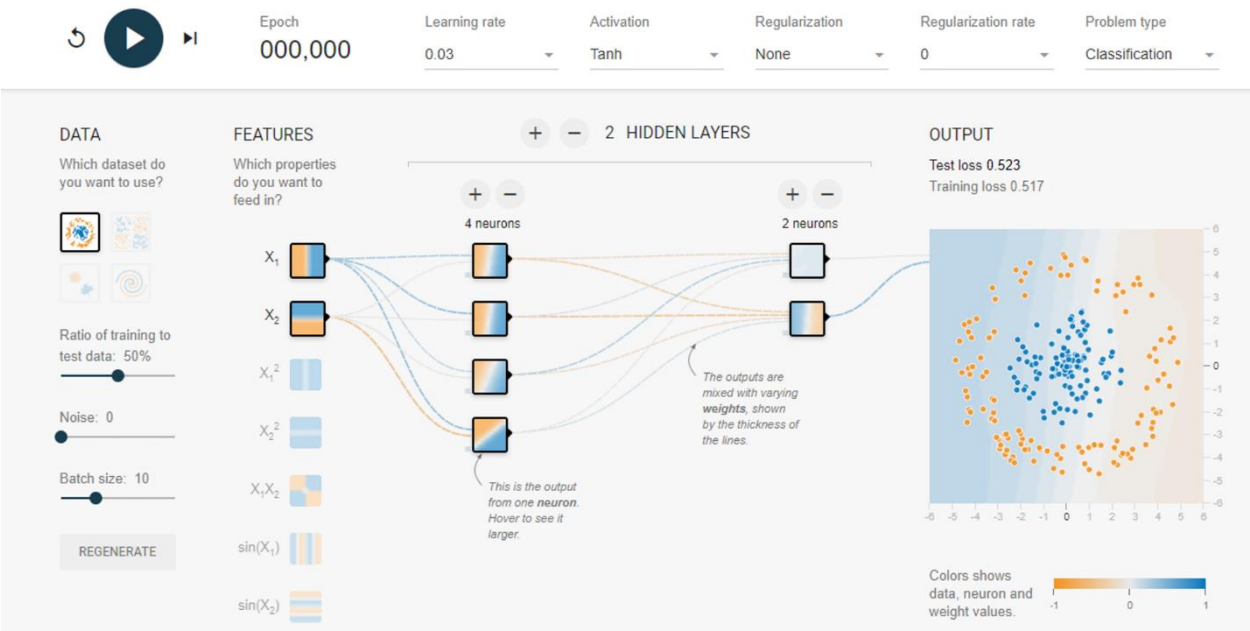

Рис.*11 TensorFlow*

Tensorflow — досить молодий фреймворк для глибокого машинного навчання, що розробляється в Google Brain. Довгий час фреймворк розроблявся в закритому режимі під назвою DistBelief, але після глобального рефакторінга 9 листопада 2015 року було випущено в open source. За рік з невеликим TF доріс до версії 1.0, знайшов інтеграцію з keras, став значно швидше і отримав підтримку мобільних платформ. Останнім часом фреймворк розвивається ще й в сторону класичних методів, і в деяких частинах інтерфейсу вже чимось нагадує scikit-learn. До поточної версії інтерфейс змінювався активно і часто, але розробники пообіцяли заморозити зміни в API. Ми будемо розглядати тільки Python API, хоча це не єдиний варіант — також існують інтерфейси для C ++ і мобільних платформ.

TF встановлюється стандартно через python pip. Є нюанс: існують окремі алгоритми установки для роботи на CPU.

У випадку з CPU все просто: потрібно поставити pip пакет під назвою tensorflow.

У другому випадку потрібно:

2. Перевірити сумісність з відеокартою. Параметр CUDA Compute Capability повинен бути більше 3.0
3. Встановити CUDA Toolkit восьмої версії
4. Встановити cuDNN версії 5.1
5. Встановити з pip пакет tensorflow-gpu

Втім, документація стверджує, що підтримуються і більш ранні версії CUDA Toolkit і cuDNN, але рекомендує встановлювати версії, зазначені вище.

Розробники рекомендують встановлювати TF в окрему середу з virtualenv, щоб уникнути можливі проблеми з версіонірованія і залежностями [4].

Ще один варіант установки — Docker. За замовчуванням з контейнера буде працювати тільки CPU-версія, але якщо використовувати спеціальний nvidia docker , то можна використовувати і GPU.

Машинне навчання Azure. Машинне навчання Azure — це хмарна служба для прискорення життєвого циклу проекту машинного навчання та управління ним. Інтерфейс зображено на рисунку 12. Фахівці з роботи з машинним навчанням, фахівці з обробки та аналізу даних та інженери даних можуть використовувати його у повсякденних робочих процесах: навчанні та розгортанні моделей, а також управлінні MLOps.

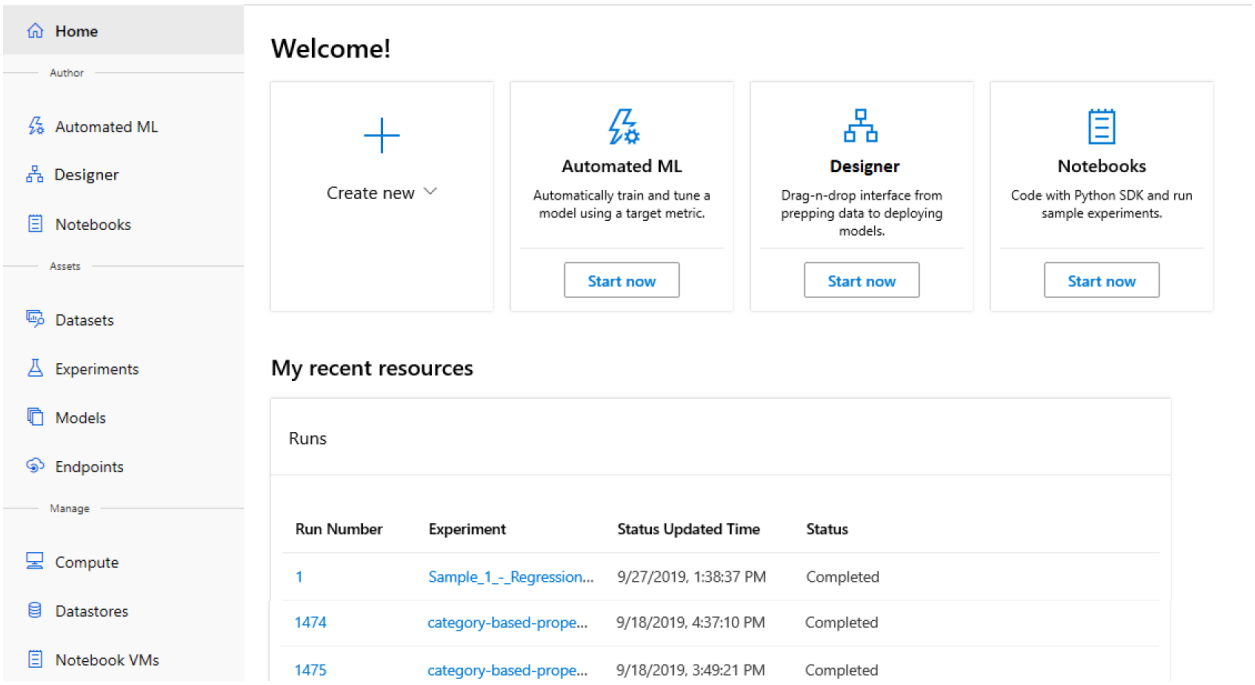

Рис.*12 Azure*

Можна створити модель у Машинному навчанні Azure або використовувати модель, побудовану на основі платформи з відкритим кодом, наприклад Pytorch, TensorFlow або scikit-learn. Засоби MLOps допомагають відстежувати, переучувати та повторно розгортати моделі.

Машинне навчання Azure містить засоби, які допомагають забезпечити спільну роботу, наприклад, наступні:

- Загальні записники, обчислювальні ресурси, дані та середовища.
- Відстеження та аудит, які показують, хто вніс зміни і коли.
- Управління версіями ресурсів.

Розробникам доступні знайомі інтерфейси в Машинному навчанні Azure, наприклад:

- Пакет SDK для Python
- REST API Azure Resource Manager.
- CLI (версія 2) – попередня версія.

Логічний процес побудови алгоритму машинного навчання:

- Визначення мети. Усі алгоритми машинного навчання нічого не варті без явно-визначеної мети проведення експерименту.
- Збір даних. Під час цього етапу формується вибірка даних, необхідна подальшого навчання моделі.
- Підготовка даних. На цьому етапі проводиться підготовка даних шляхом формування характеристик, видалення викидів та поділу вибірки на навчальну та тестову.
- Розробка моделі. У процесі розробки моделі проводиться вибір одного або кількох моделей даних та відповідних алгоритмів навчання, які на думку розробника повинні будуть дати необхідний результат. Часто цей процес поєднаний з паралельним дослідженням ефективності кількох моделей та візуальним аналізом даних з метою відшукання будь-яких закономірностей.
- Вивчення моделі. Під час навчання алгоритм навчання здійснює пошук прихованих закономірностей у вибірці даних з метою відшукання способу передбачення. Сам процес пошуку визначається обраною моделлю та алгоритмом навчання.
- Оцінка моделі. Після того, як модель навчена необхідно досліджувати її прогностичні характеристики. Найчастіше для цього її пропускають на тестовій вибірці і оцінюють рівень помилки, що вийшов. Залежно від цього і вимог до точності модель може бути прийнята як підсумкова, так і проведено повторне навчання після додавання нових вхідних характеристик або навіть зміни алгоритму навчання.
- Використання моделі. У разі успішного тестування навченої моделі настає стадія її використання. І це той випадок, коли Azure ML стає незамінним, даючи всі необхідні інструменти для публікації, моніторингу та монетизації алгоритмів [10].

Робоча область — це централізоване місце розташування, в якому можна:

- керувати ресурсами, що використовуються для навчання та розгортання моделей, такими як обчислення;
- зберігати ресурси, створені під час використання Машинного навчання Azure, у тому числі такі:
  - Середовища
  - Експерименти
  - Конвеєри
  - Набори даних
  - Моделі
  - Кінцеві точки

Інтерфейс робочої області надано на рисунку 13.

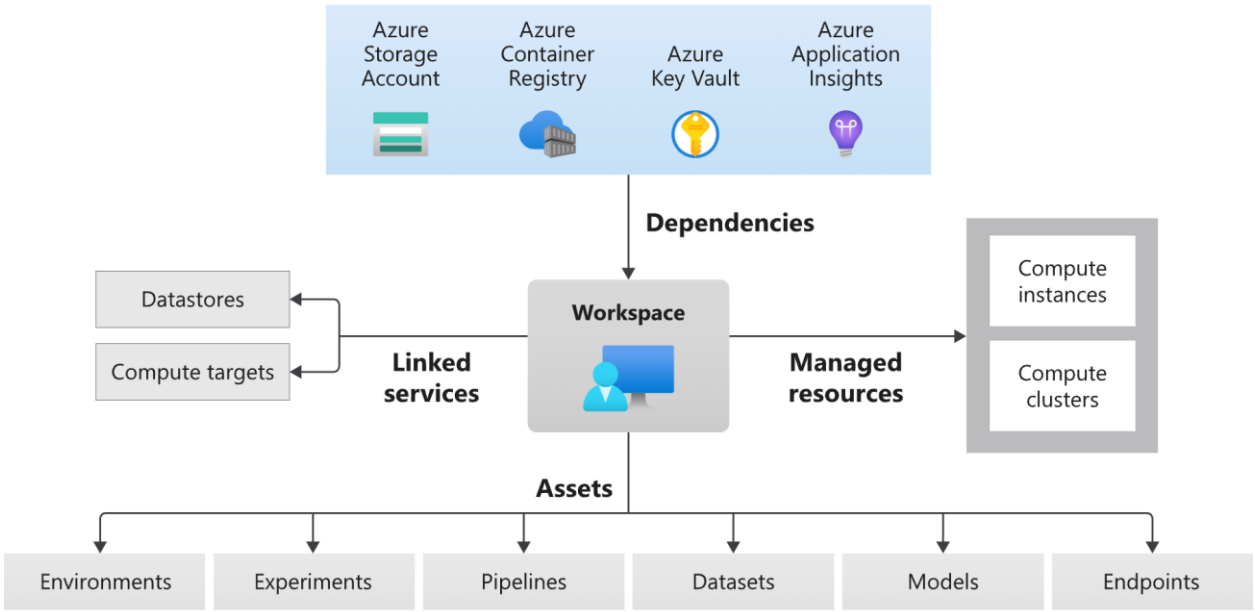

Рис.*13 Робоча область Azure*

Робоча область включає інші ресурси Azure, які використовуються робочою областю:

- Реєстр контейнерів Azure: реєструє контейнери Docker, які використовуються під час навчання та розгортання моделі. Щоб зменшити витрати, реєстр ACR створюється лише при створенні образів розгортання.
- Обліковий запис служби сховища Azure: використовується як сховища даних за промовчанням для робочої області. У ній також зберігаються записники Jupyter, які використовуються в поєднанні з обчислювальними екземплярами Машинного навчання Azure.
- Azure Application Insights: зберігає інформацію про моніторинг ваших моделей.
- Azure Key Vault: містить секрети, що використовуються цільовими об'єктами обчислень, та інші конфіденційні відомості, необхідні робочій області.

Amazon SageMaker. Amazon SageMaker — це повністю керована служба машинного навчання. За допомогою SageMaker науковці та розробники даних можуть швидко й легко створювати й навчати моделі машинного навчання, а потім безпосередньо розгортати їх у готовому для виробництва середовищі.

Він надає інтегрований екземпляр блокнота для розробки Jupyter для легкого доступу до джерел даних для дослідження та аналізу, тому вам не потрібно керувати серверами.[14]

Він також надає загальні алгоритми машинного навчання, які оптимізовані для ефективної роботи з надзвичайно великими даними в розподіленому середовищі. Завдяки вбудованій підтримці алгоритмів і фреймворків «принесіть свої власні», SageMaker пропонує гнучкі розподілені варіанти навчання, які адаптуються до ваших конкретних робочих процесів.

Навчання та хостинг оплачуються за хвилини використання, без мінімальних платежів і авансових зобов’язань.

Amazon SageMaker включає такі функції:

- Студія SageMaker. Інтегроване середовище машинного навчання, де можна створювати, навчати, розгортати й аналізувати моделі в одній програмі.
- Реєстр моделей SageMaker. Відстеження версій, відстеження артефактів і походження, робочий процес схвалення та підтримка між обліковими записами для розгортання моделей машинного навчання.
- Проекти SageMaker. Створення наскрізних рішеннь ML із CI/CD за допомогою проектів SageMaker.
- Конвеєри для створення моделей SageMaker. Створення та керування конвеєрами машинного навчання, інтегрованими безпосередньо із завданнями SageMaker.
- Налагоджувач SageMaker. Перевірка параметрів навчання та даних протягом усього навчального процесу. Автоматично виявлення й сповіщання користувачів про поширені помилки, наприклад, занадто великі чи малі значення параметрів.
- Навчання з підкріпленням. Максимізування довгострокої винагороди, яку отримує агент в результаті своїх дій [11].

РОЗДІЛ 3 РОЗРОБКА ДОДАТКУ ДЛЯ КЕРУВАННЯ СОЦІАЛЬНИМИ ГРУПАМИ ЗА ДОПОМОГОЮ НЕЙРОННОЇ МЕРЕЖІ.

3.1 Вимоги до апаратного та програмного забезпечення

Мінімальні вимоги до апаратного та програмного забезпечення:

- Операційна система: Windows 7, macOs 15+;
- Оперативна пам’ять: 6 Гб;
- Процесор: Core Intel або AMD або M1.
- Доступ до Інтернету

Рекомендовані вимоги до апаратного та програмного забезпечення:

- Операційна система: MacOS Big Sur
- Оперативна пам’ять: 16 Гб;
- Процесор: M1 та вище;

3.2 Засоби реалізації

Python

Засобом реалізації алгоритму Q була обрана мова програмування Python. Проведений аналіз досвіду людей при роботі з нейронними мережами переважно використовують саме цю мову для розробки нейронної мережі. Крім простоти, лаконічності та виразності, що дозволяють з мінімальними витратами часу та сил розробляти складні алгоритми, мова має ще й потужний механізм інтероперабельності з C\C++, що відкриває доступ до швидких обчислень [13].

Наприклад, у ньому не потрібно вказувати тип даних, досить просто оголосити змінну. З контексту Python зрозуміє, чи є вона цілим числом, числом з плаваючою комою, логічним значенням чи чимось іншим. Це величезна перевага для початківців.

Для реалізації нейронної мережі потрібно створити моделі агентів та поточного стану, після чого нейронна мережа буде навчати агентів виживати у середовищі до якого ми їх помістили.

Для вирішення поставленої задачі було обрано Q алгоритм завдяки його простоті та ефективності. Після того як нейронна мережа відпрацює всі кроки навчання та знайде рішення вона збереже процес навчання у JSON форматі. Кожний стан та кожну подію можливо буде продемонструвати використовуючи пристосоване до цього формату даних середовище.

Dart

Dart — це об’єкно-орієнтована мова програмування, розроблена компанією Google та призначена перш за все для розробки веб-застосунків та додатків під мобільні телефони. Одну й ту ж програму можна компілювати під різні платформи [6].

Набір основних бібліотек:

- dart:core: вбудовані типи, колекції та інші основні функції;
- dart:collection: типи колекцій, як черги, пов'язані списки, хеш-карти та двійкові дерева;
- dart:convert: кодери та декодери для перетворення між різними представленнями даних;
- dart:math: математичні константи та функції, рандом;
- dart:io підтримка файлів, сокетів, HTTP та інших операцій;
- dart:async: підтримка асинхронного програмування;
- dart:typed_data: списки, які ефективно обробляють дані фіксованого розміру;
- dart:ffi: інтерфейси зовнішніх функцій для сумісності з іншим кодом, що представляють інтерфейс у стилі C;
- dart:isolate: паралельне програмування з використанням ізолятів;

- dart:html: HTML-елементи та інші ресурси для веб-додатків, яким необхідно взаємодіяти з браузером та об'єктною моделлю документа (DOM).

Компілятор Dart дозволяє запускати такими способами (Див.рис.14):

- Native platform: віртуальна машина Dart зі своєчастною (just-in-time) та завчастною (ahead-of-time) компіляцією для мобільних та настільних додатків
- Web platform: компілятор часу розробки (dartdevc) та компілятор часу виробництва (dart2js) для веб-застосунків, що переводять Dart в JavaScript.

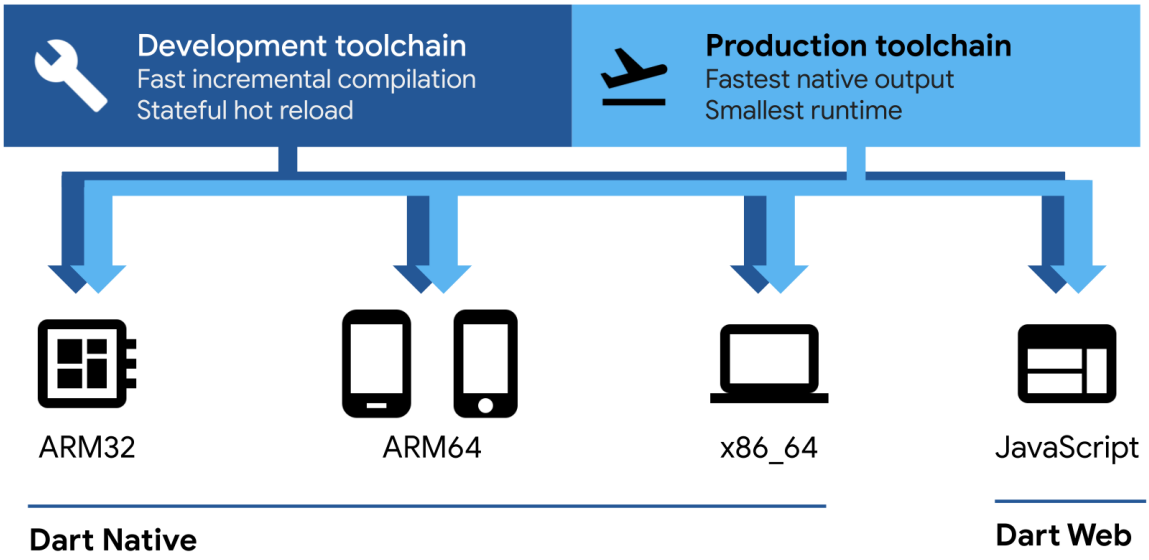

Рис.*14 Компілятор Dart*

Flutter

Для реалізації середовища було обрано фреймворк для кросплатформної розробки Flutter. Даний фреймворк відрізняється від інших своєю простотою та швидкістю не тільки роботі самої програми але і швидкістю розробки. Вся робота за користувацьким інтерфейсом пов’язана на декларативному програмуванні інтерфейсу, даний фреймворк використовує віджети як осно-

вний компонент користувацького інтерфейсу. Користь даного фреймворку в тому що він може мати одну базу коду та завдяки 5 командам можна створити додатки під будь яку платформу будь то web, iOS, Android, Windows, macOS чи Linux.

Flutter — це фреймвор з відкритим вихідним кодом для створення вебзастосунків та мобільних додатків на мові Dart. Flutter розроблений компанією Google [7].

Основні складові фреймворку:

- платформа Dart;
- двигун Flutter;
- бібліотека Foundation;
- набори віджетів;
- засоби розробки (Flutter DevTools).

При побудові програми Flutter транслює Dart код у нативний код програми за допомогою Dart AOT (компіляція програми перед його запуском), але для прискорення робробки Flutter використовує JIT (компіляція програми під час його запуску).

Інтерфейс програм на Flutter передбачає використання віджетів. Усі графічні об’єкти: текст, форми та анімація, створюються за допомогою віджетів та їх комбінування. Фреймворк надає два основні набори віджетів — Material Design (стиль Google) та Cupertino (стиль Apple).Створювати програми на Flutter можна і без віджетів, викликаючи методи бібліотеки Foundation для роботи з канвою [12].

Bloc

Block — це бібліотека управління станом, що допомагає реалізувати шаблон проектування BLoC для спрощення розділу представлення від бізнес-

логіки, полегшуючи тестування та можливість повторного використання [8]. Схема представлена на рисунку 15.

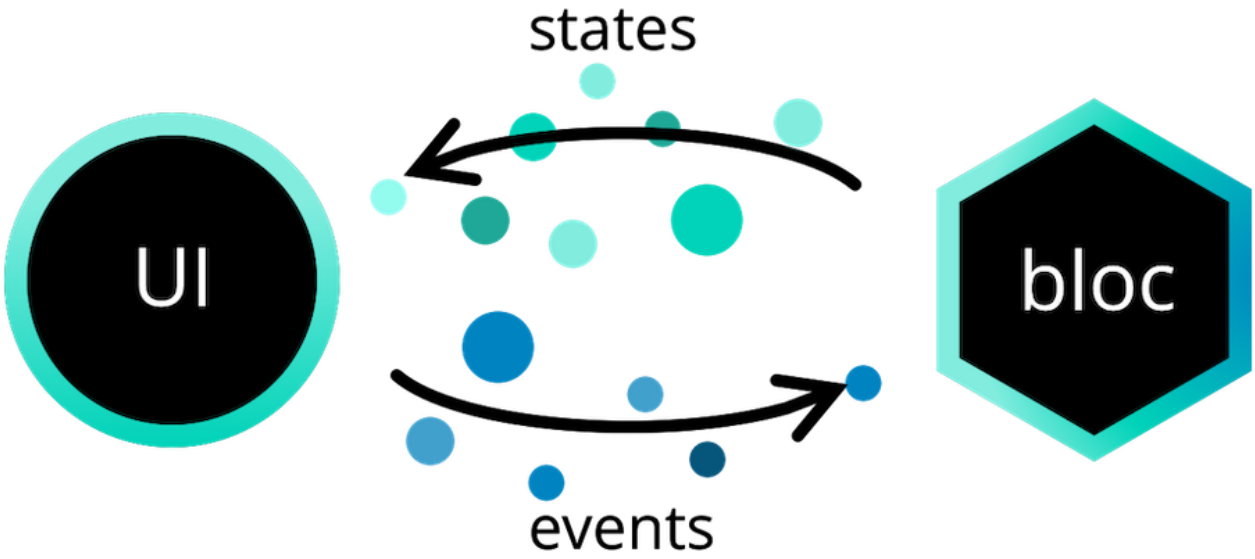

Рис.*15 Bloc*

Блок складається з кількох паб-пакетів:

- bloc — бібліотека основного блоку;
- flutter_bloc — віджети Flutter, створені для роботи з bloc, щоб створювати швидкі, реактивні мобільні додатки;
- angular_bloc —компоненти Angular, створені для роботи з bloc для створення швидких, реактивних веб-додатків;
- hydrated_bloc — розширення бібліотеки управління станом блоку, яке автоматично зберігається та відновлює стани блоку;
- replay_bloc — розширення бібліотеки управління станом блоку, яке додає підтримку скасування та повторного виконання.

3.3Класи та методи

MyHomePage. Клас для для головногу екрану додадка.

На цьому екрані буде дві секції одна секція відповідає за анімоване відображення роботі нейронної мережі, а друга секція для відображення поточного стану кожного з агентів.

Лістинг *1 Клас* **MyHomePage**

```
class MyApp extends StatelessWidget {
const MyApp({Key? key}) : super(key: key);

@override
Widget build(BuildContext context) {
return MaterialApp(
title: 'Flutter Demo',
theme: ThemeData(
primarySwatch: Colors.grey,
),
home: const MyHomePage(title: ''),
);
}
}

class MyHomePage extends StatefulWidget {
const MyHomePage({Key? key, required this.title}) : super(key: key);
final String title;

@override
State<MyHomePage> createState() => _MyHomePageState();
}

class _MyHomePageState extends State<MyHomePage> {
@override
```

```
Widget build(BuildContext context) {
return Scaffold(
backgroundColor: Color(0xff202020),
body: Center(
child: Row(
children: [
Expanded(
flex: 3,
child: Padding(
padding: const EdgeInsets.all(8.0),
child: Container(
color: const Color(0xff202020),
child: Column(
children: [
const SizedBox(
height: 50,
child: Center(
child: Text(
'Area',
style: TextStyle(color: Colors.grey, fontSize: 25),
),
),
),
Expanded(
child: Container(
height: double.infinity,
width: double.infinity,
decoration: BoxDecoration(
color: const Color(0xff355e3b),
borderRadius: const BorderRadius.only(
topLeft: Radius.circular(10),
topRight: Radius.circular(10),
```

```
bottomLeft: Radius.circular(10),
bottomRight: Radius.circular(10)),
boxShadow: [
BoxShadow(
color: Colors.black.withOpacity(0.6),
spreadRadius: 5,
blurRadius: 5,
offset:
Offset(0, 3), // changes position of shadow
),
],
),
child: LayoutBuilder(
builder: (BuildContext context,
BoxConstraints constraints) {
Random random = Random();
return Stack(children: [
...agents
.map(
(a) => AnimatedPositioned(
left: a.position.x,
top: a.position.y,
child: Image.asset(
a.getAssetPath(),
height: 70,
width: 70,
),
duration:
const Duration(milliseconds: 400),
),
)
.toList(),
```

```
...resources
.map(
(r) => AnimatedPositioned(
left: 100.0 + random.nextInt(720),
top: 100.0 + random.nextInt(700),
child: Image.asset(
r.getAssetPath(),
height: 70,
width: 70,
),
duration:
const Duration(milliseconds: 400),
),
)
.toList(),
]);
},
),
),
),
],
),
),
),
),
Expanded(
child: Container(
color: const Color(0xff202020),
child: Padding(
padding: const EdgeInsets.only(top: 58.0),
child: Column(
crossAxisAlignment: CrossAxisAlignment.start,
```

```
children: [
const Text(
'Current State:',
style: TextStyle(color: Colors.grey, fontSize: 25),
),
...agents.map(
(e) => Padding(
padding: const EdgeInsets.only(left: 28.0, top: 20),
child: Column(
crossAxisAlignment: CrossAxisAlignment.start,
children: [
Row(
children: [
Image.asset(
e.getAssetPath(),
height: 30,
width: 30,
),
Text(
e.type
.toString()
.split('.')[1]
.toUpperCase(),
style: const TextStyle(
color: Colors.grey, fontSize: 25),
),
],
),
Padding(
padding: const EdgeInsets.only(left: 71.0),
child: Column(
crossAxisAlignment: CrossAxisAlignment.start,
```

```
children: [
...e.resources
.map(
(r) => Row(
children: [
Image.asset(
r.getAssetPath(),
height: 30,
width: 30,
),
Text(
': ${r.count}',
style: const TextStyle(
color: Colors.grey,
fontSize: 25),
),
],
),
)
.toList(),
],
),
),
],
```

Q_learning. Даний клас реалізує роботу самого алгоритму він приймає та як вхідне дане агентів, кількість епізодів та Q-table. Та після обробки повертає оновлену Q-table. Завдяки котрій агенти в подальшому зможуть виживати у даному середовищі.

Лістинг *2 Клас* **Q_learning**

```
class Agent:

def __init__(self):
    self.states = []
    self.actions = [0, 1, 2, 3]
    self.MyState = State()
    self.alpha = 0.9
    self.gamma = 0.9
    self.epsilon = 0.5
    self.isEnd = self.MyState.isEnd

    self.plot_reward = []

    self.Qtable = {}
    self.new_Q = {}
    self.myrewards = 0

    for i in range(BOARD_ROWS):
        for j in range(BOARD_COLS):
            for k in range(len(self.actions)):
                self.Qtable[(i, j, k)] = 0
                self.new_Q[(i, j, k)] = 0

    print(self.Qtable)
    print('\n')

def Action(self,episodes):
    rnd = random.random()
    mx_nxt_reward = -10
    action = None
    if (rnd > self.epsilon or episodes>350):
        for k in self.actions:
```

```
                i, j = self.MyState.state

                nxt_reward = self.Qtable[(i, j, k)]

                if nxt_reward >= mx_nxt_reward:
                    action = k
                    mx_nxt_reward = nxt_reward

        else:
            action = np.random.choice(self.actions)

        position = self.MyState.nxtPosition(action)

        return position, action

    def Q_Learning(self, episodes):
        x = 0
        while (x < episodes):
            if self.isEnd:
                reward = self.MyState.getReward()
                self.myrewards += reward
                self.plot_reward.append(self.myrewards)

                i, j = self.MyState.state
                for a in self.actions:
                    self.new_Q[(i, j, a)] = round(reward,
3)

                self.MyState = State()
                self.isEnd = self.MyState.isEnd
```

```
                self.myrewards = 0
                x += 1
            else:
                mx_nxt_value = -10
                nextstate, action = self.Action(x)
                i, j = self.MyState.state
                reward = self.MyState.getReward()
                self.myrewards += reward

                for a in self.actions:
                    nxtStateAction = (nextstate[0],
nextstate[1], a)
                    q_value = (1 - self.alpha) *
self.Qtable[(i, j, action)] + self.alpha * (
                                reward + self.gamma *
self.Qtable[nxtStateAction])

                    if q_value >= mx_nxt_value:
                        mx_nxt_value = q_value

                self.MyState = State(state=nextstate)
                self.MyState.isEndFunc()
                self.isEnd = self.MyState.isEnd

                self.new_Q[(i, j, action)] =
round(mx_nxt_value, 3)

            self.Qtable = self.new_Q.copy()
        print(self.Qtable)
        print('\n')
```

```
    def plot(self, episodes):

        plt.plot(self.plot_reward)
        plt.show()


    def showValues(self):
        for i in range(0, BOARD_ROWS):
            print('-----------------------------------
-------'*5)
            out = '| '
            for j in range(0, BOARD_COLS):
                mx_nxt_value = -10
                for a in self.actions:
                    nxt_value = self.Qtable[(i, j, a)]
                    if nxt_value >= mx_nxt_value:
                        mx_nxt_value = nxt_value
                out += str(mx_nxt_value).ljust(6) + ' | '
            print(out)
```

Agent. Даний клас реалізує модель агента у середовищі відображення. Та в залежності від значень його полів демонструє у застосунку певний стан.

Даний класс також має метод який повертає шлях до картинки певного агенту.

Лістинг *3 Клас* **Agent**

```
import 'package:ai_demo/position.dart';
import 'package:ai_demo/resource.dart';

enum AgentType { lumberjack, builder, fisherman,
medecineman }
```

```
class Agent {
AgentType type;
List<Resource> resources;
Position position;
Agent({
required this.position,
required this.resources,
required this.type,
});

String getAssetPath() {
return 'assets/${type.toString().split('.')[1]}.png';
}
}
```

Resource. Даний клас реалізує модель ресурси у середовищі відображення. Та в залежності від значень його полів демонструє у застосунку певний стан.

Даний класс також має метод який повертає шлях до картинки певного ресурсу.

Лістинг *4 Клас* **Resource**

```
enum ResouorceType { stone, fish, flowers, tree }

class Resource {
ResouorceType type;
int count;
Resource({
required this.type,
required this.count,
```

```
});

String getAssetPath() {
return 'assets/${type.toString().split('.')[1]}.png';
}
}
```

Pubspec.yaml. данний файл допомагае підключити необхідні пакети до нашого проекту, які в подальшому можуть спростити нашу роботу.

Лістинг *5 Файл* **Pubspec.yaml**

```
version: 1.0.0+1
environment:
sdk: ">=2.12.0 <3.0.0"
dependencies:
flutter:
sdk: flutter
cupertino_icons: ^1.0.2
dev_dependencies:
flutter_test:
sdk: flutter
flutter_lints: ^1.0.0
flutter:
uses-material-design: true
assets:
- assets/
```

3.4 Створення оточення

Оточення для навчання складається з трьох компонентів. По-перше це агенти які будуть навчатись, по-друге це карта на якій будуть пересуватись самі агенти та по-третє це ресурси з якими будуть взаємодіяти агенти.

Для початку розглянемо всіх агентів, їх всього 4 види:

- Рибаки
- Будівельники
- Лісоруби
- Лікарі

Агенти Рибаки зображені на рисунку 16.

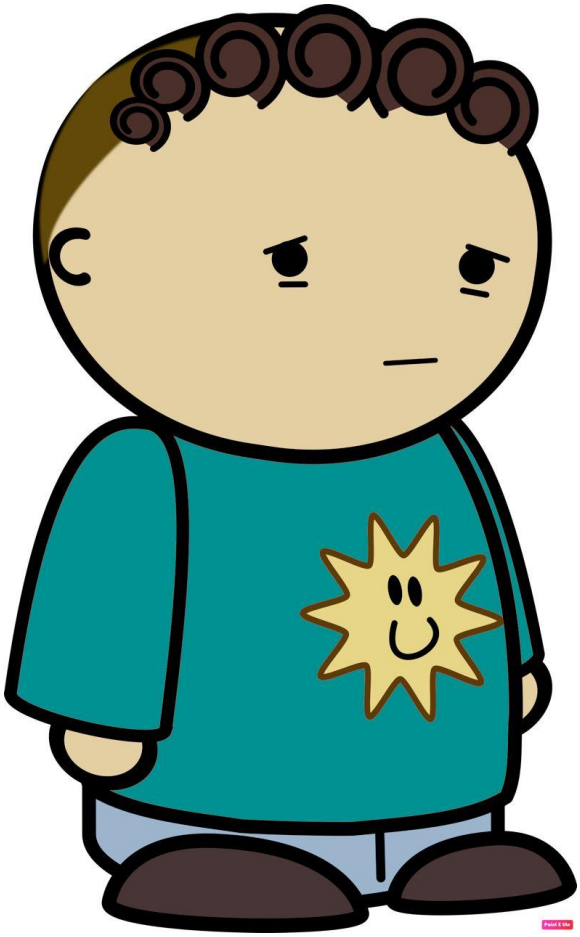

Рис.*16 Рибаки*

Дана група агентів займається ловлею риби.

Переваги:

- Можуть ловити рибу.
- Не помирають від голоду.
- Вміють плавати.

Недоліки:

- Не мають власного будинку.
- Не можуть пережити хворобу.
- Помирають від холоду.

Дана група агентів займається ловлею риби, тому вони завжди мають їжу та голод для них не може бути проблемою. Але, за відсутністю житла, стихійне лихе призводить до загибелі всієї групи. Через те, що дана група немає ліків, хвороба також може призвести до вимирання. Крім холоду дану групу агентів може знищити мороз, бо вони не мають дров аби зігрітись.

Агенти Лісоруби зображені на рисунку 17.

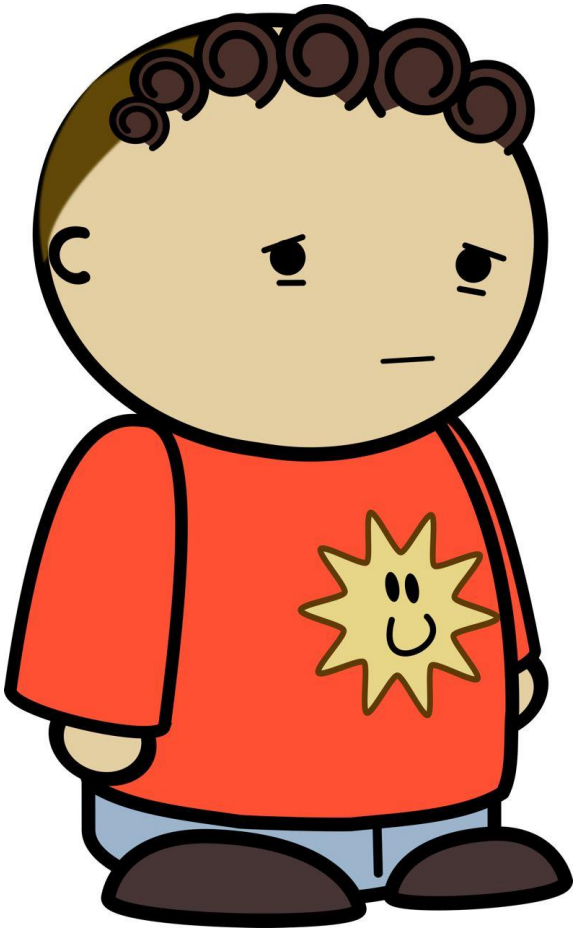

Рис.*17 Лісоруби*

Дана група агентів займається вирубкою лісу.

Переваги:

- Можуть вирубати ліс.
- Можуть розпалювати багаття.

Недоліки:

- Не мають власного будинку.
- Не можуть пережити хворобу.
- Помирають від голоду.

Дана група агентів рятує себе від заморозків за рахунок того, що нарубає достатню кількість дров та може розпалити багаття, яке врятує їх від смерті через обмороження. Але за недоліком їжі помирає від голоду. При наступному стихійного лиха не мають будинків щоб сховатися, що також в свою

чергу призводить до вимирання, а через те, що вони не мають ліків, хвороба також не обійде їх стороною.

Агенти Будівельники зображені на рисунку 18.

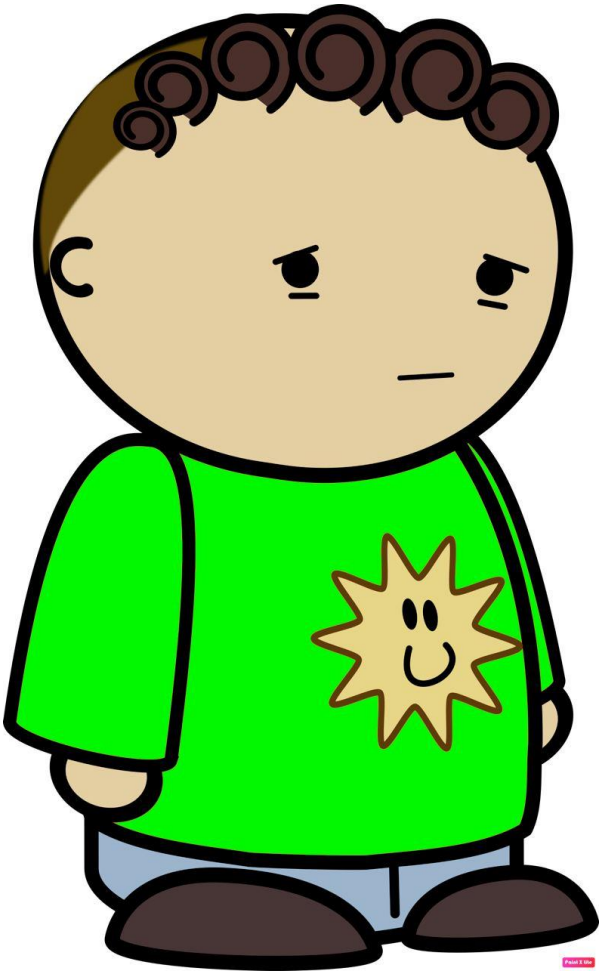

Рис. *18 Будівельники*

Дана група агентів займається будівництвом.

Переваги:

- Мають своє житло.
- Вміють будувати житло.

Недоліки:

- Не можуть пережити хворобу.
- Помирають від голоду.

Дана група агентів займається побудовою будинків тему стихійні лиха для них не проблема, але через невміння розпалювати багаття холод може призвести до вимирання даного виду, через недолік лік хвороба також призводить до вимирання. Через відсутність їжі голод також призводить до вимирання даного виду агентів.

Агенти Лікарі зображені на рисунку 19.

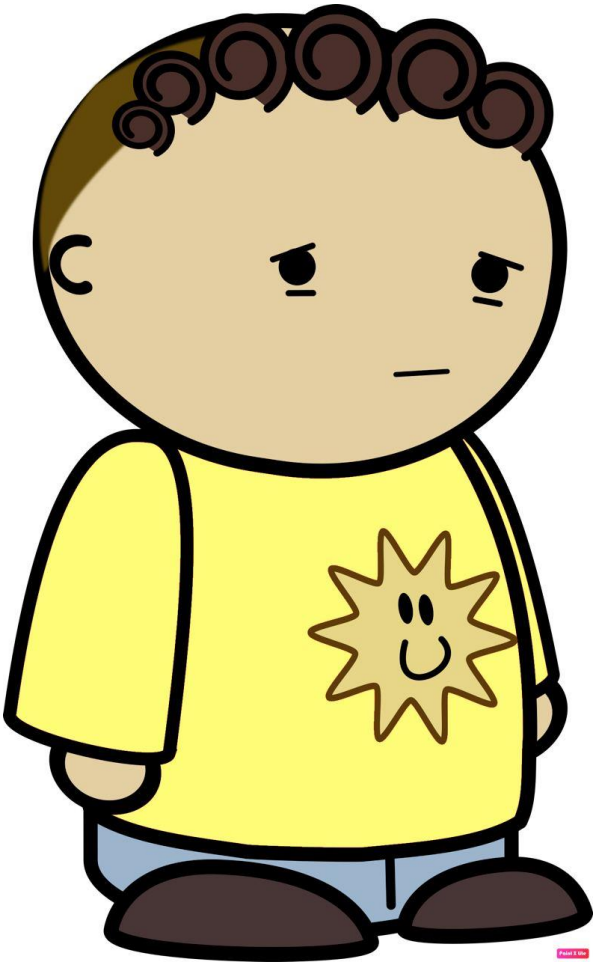

Рис. *19 Лікарі*

Дана група агентів займається збором лікарських трав.

Переваги:

- Можуть збирати лікарські трави.
- Можуть створити ліки від хвороби.

Недоліки:

- Не мають власного будинку.
- Не можуть пережити холод.
- Помирають від голоду.

Дана група агентів займається збором та створенням лікарських трав. Нехватка їжі призводить до голоду, який в свою чергу призводить до вимирання, відсутність дров та будинку також негативно позначається на даній групі агентів.

Ресурси

Данна програма має 4 види ресурсів:

- каміння;
- дерева;
- трави;
- риба.

Каміння використовують будівельники та збирати каміння можуть тільки будівельники. Зображення каміння продемонстровано на малюнку 20.

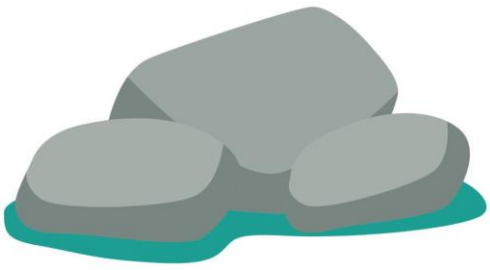

Рис. *20 Каміння*

Дерева використовують лісники та збирати дрова можуть тільки лісники. Зображення дерева продемонстровано на малюнку 21.

Рис. *21 Дерево*

Трави використовують лікарі та збирати трави можуть тільки лікарі. Зображення трав продемонстровано на малюнку 22.

Рис. *22 Трава*

Риби використовують рибаки та ловити рибу можуть тільки рибаки. Зображення риби продемонстровано на малюнку 23.

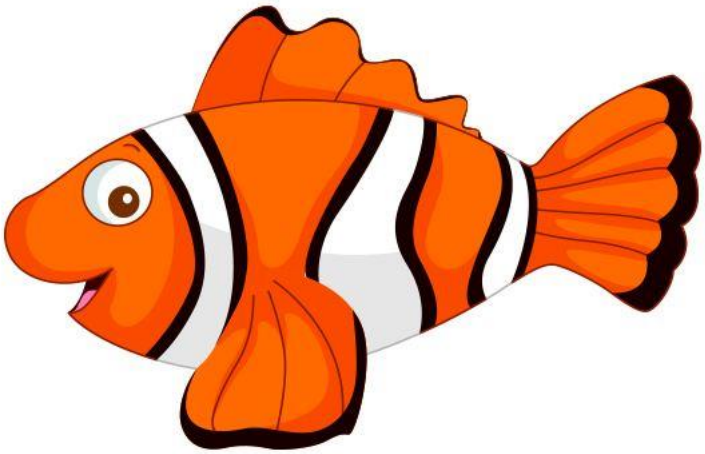

Рис. *23 Риба*

В процес життєдіяльності агентів було включено 4 стихійних лиха.

- голод;

- заморозки;
- торнадо;
- хвороба.

У симуляції буде реалізоване поступове появлення того чи іншого стихійного лиха.

Додаток має одну сцену, на котрій будуть усі чотири агента, які зображені на рисунку 24. Кожен агент у своєму кутку карти поряд з ресурсами, на які у даного агента є дозвіл на використання.

Рис. *24 Карта*

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РОБОТИ НЕЙРОННОЇ МЕРЕЖІ

4.1 Q-learning

Q-навчання — це простий спосіб для агентів навчитися оптимально діяти в контрольованих областях Маркова. Це є інкрементним методом для динамічного програмування, який накладає обмежені обчислювальні вимоги. Він працює шляхом послідовного покращення оцінок якості конкретних дій у певних державах [9].

Q-навчання є формою навчання з підкріпленням без моделі. Його також можна розглядати як метод асинхронного динамічного програмування. Він надає агентам можливість навчитися оптимально діяти в доменах Маркова, відчуваючи наслідки дій, не вимагаючи від них побудови карт доменів.

Навчання відбувається подібно до методу часових різниць Саттона: агент намагається виконати дію в певному стані і оцінює його наслідки з точки зору негайної винагороди або покарання, яку він отримує, і своєї оцінки вартості стан, до якого воно доставлено. Неодноразово випробовуючи всі дії у всіх станах, він дізнається, які в цілому найкращі, судячи з накопичення нагород.

Процес прийняття рішень Маркова описує середовище для навчання з підкріпленням, в якому поточний стан показує нам усе, що нам потрібно знати про майбутні стани. Це означає, якщо ми знаємо поточний стан навколишнього середовища в процесі прийняття рішень Маркова, нам не потрібно нічого знати про будь-які попередні стани, щоб визначити, якими будуть майбутні стани, або вирішити, які дії вжити в цьому поточному стані [8].

Схему можна представити у вигляді кінцевого автомату, зображеного на рисунку 25. Стани позначені S_0, S_1 та S_2. Коли ми перебуваємо в стані S_0, ми можемо зробити будь-яку з цих дій a_0 або дію a_1. Якщо ми вживаємо заходів a_0, є 50% шанс, що ми опинимося в штаті S_2 і 50% ймовірність того, що

ми повернемося до штату S_0. Якщо ми вживаємо заходів a_1, є 100% шанс, що ми про кінцевий результат поринемо в штат S_2 і так далі.

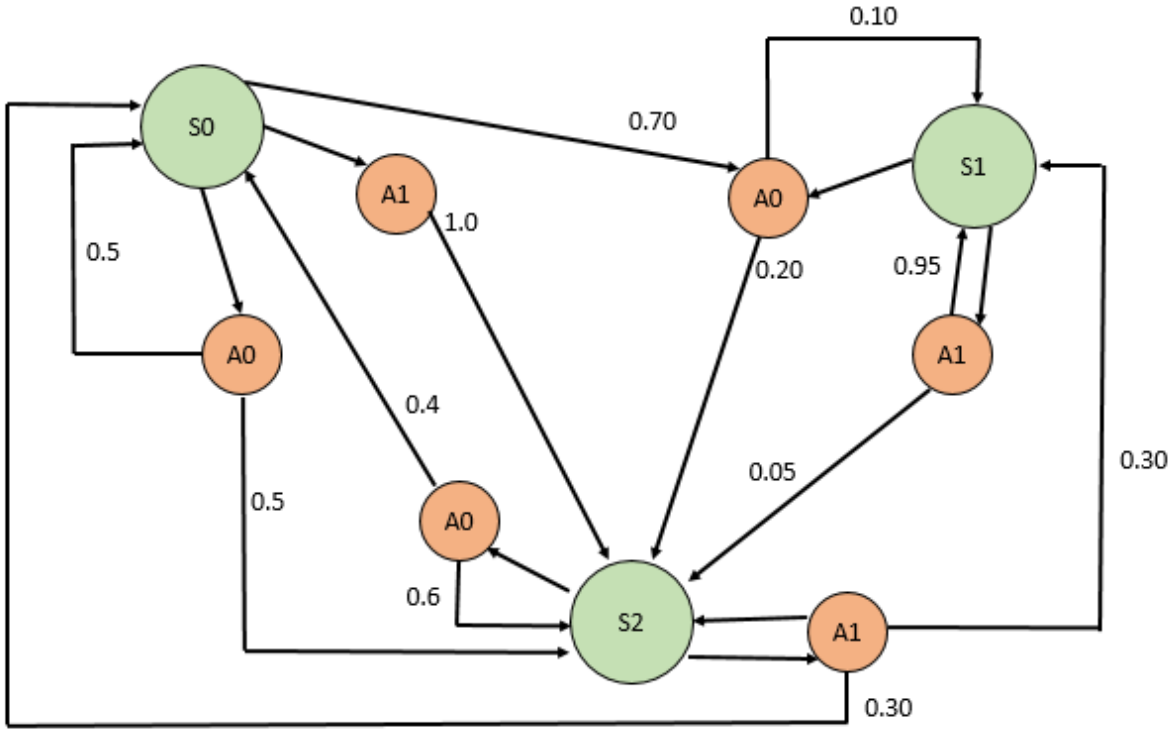

Рис.*25 Кінцевий автомат*

Різні дії, які ми вибираємо, можуть мати різні можливі результати, які ми визначимо згодом з допомогою спостереження. У процес прийняття рішень Маркова, з якими ми працюємо, будуть нагороди, пов'язані з кожним кроком, і наша мета буде максимізувати нагороди, які ми отримуємо, знаючи результати кожної дії, яку ми обираємо. Чим більше ми дізнаємося про це середовище з часом, тим краще становище, в яке ми знаходимося, щоб робити дії з високою винагородою, яку ми бачили раніше.

Марківський ланцюг. Марківський ланцюг — це модель системи подій, де ймовірність наступної події залежить тільки від стану попередньої (або поточної події). Марківський ланцюг зображено на рисунку 26.

Рис.*26 Марківський ланцюг*

Ланцюг має два стани, і система рухається між ними. Стрілки між ними — це ймовірність, що ми перейдемо у відповідні стани. Якщо ми перебуваємо в стані Win, є 60% ймовірність того, що ми залишимося у Win і 40% ймовірність того, що ми перейдемо в Lose на наступному часовому кроці (час, необхідний для гри в ще один раунд). Так само, якщо ми програємо цей раунд, є 70% шанс, що ми виграємо наступний раунд, і 30% шанс, що ми знову програємо.

4.2 Налаштування параметрів нейронної мережі

Щоб зрозуміти усі параметри нейронної мережі потрібно протестувати її міняючи їх. Для початку потрібно розуміти які параметри будемо досліджувати. В даній роботі можна спостерігати три основних параметри:

- альфа;
- гамма;
- епсілон.

Епсілон — це показник випадковості для прийняття рішення агентом. Він існує за для того щоб агенті не зациклювались на одному місці коли змогли отримати винагороду. Простіше кажучи цей показник відповідає за те з

якою вірогідністю агент буде обирати випадковий шлях, який в подальшому може призвести до більшої винагороди.

Тепер наведемо приклад на основі нашої нейронної мережі. Для початку зробимо цей показник 0 та подивимось як буде проходити навчання нейронної мережі на графіку, зображеному на рисунку 27.

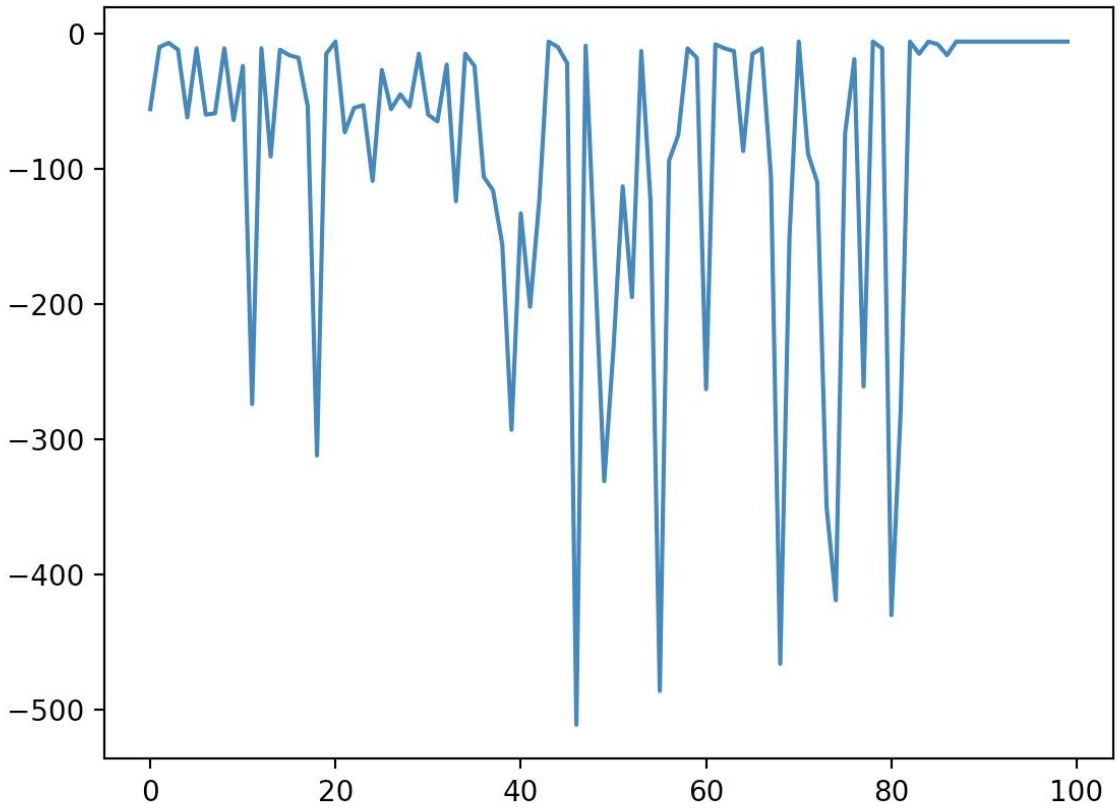

Рис.*27 Навчання*

Як можна побачити нейронна мережа не вчиться, тому що завжди обирає найвигіднішу позицію.

Тепер змінимо цей показник до одиниці та подивимося, що змінилося на рисунку 28.

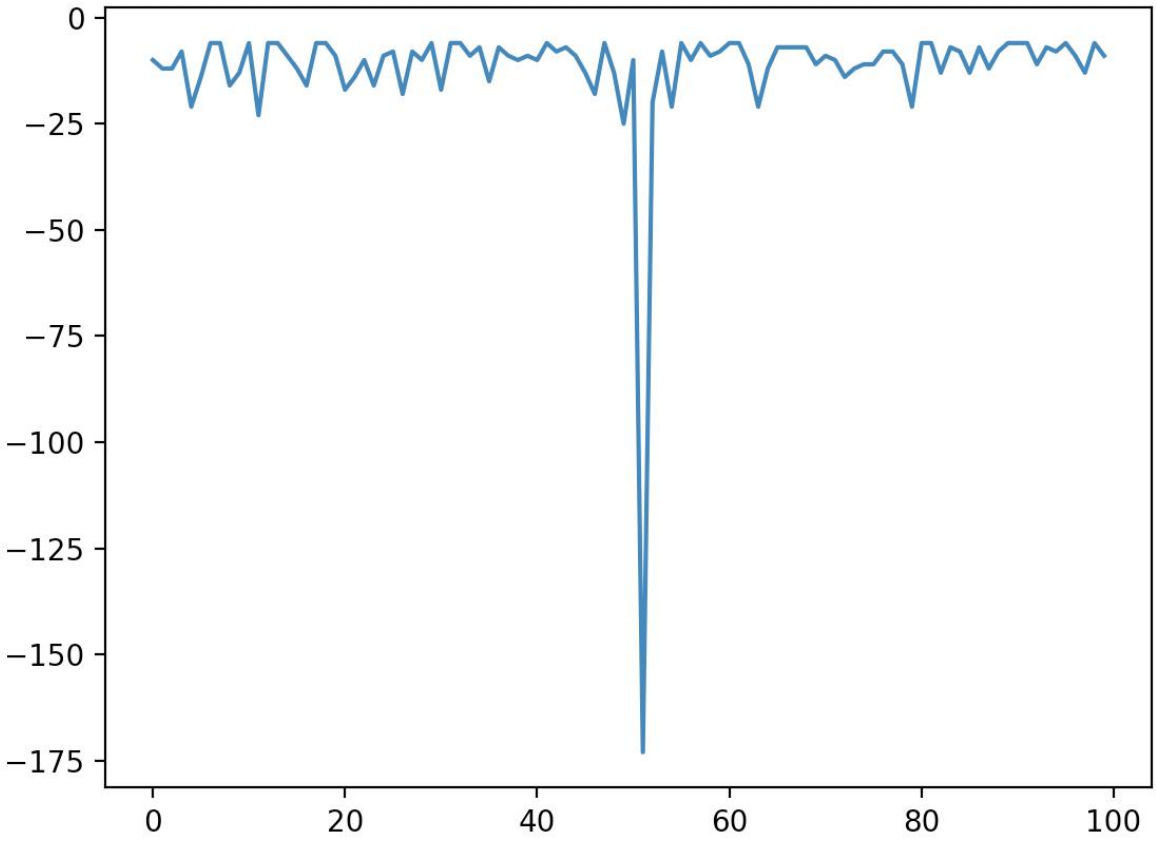

Рис.*28 Навчання*

Отже також можна побачити який сильний розбіг у кінцевій винагороді ми отримали.

Тепер зробимо цей показник 0,5 щоб агент за вірогідністю приймав випадкове рішення, результат зображено на рисунку 29.

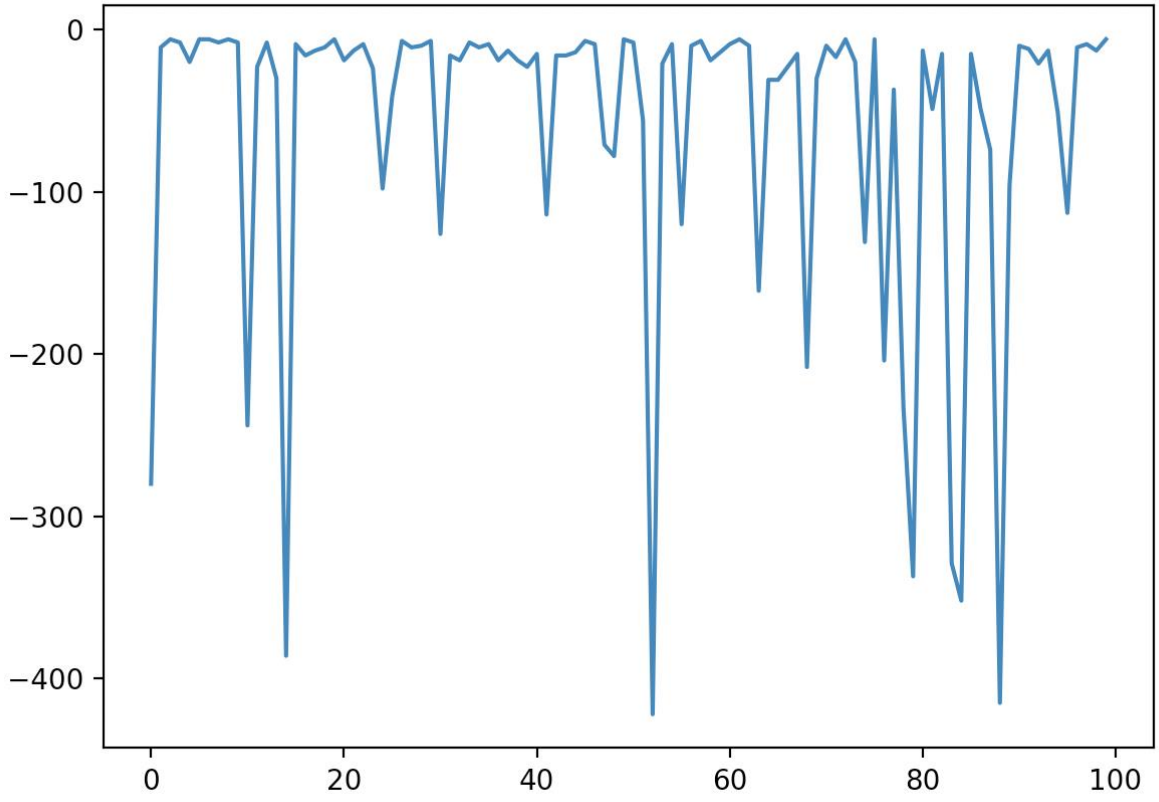

Рис. *29 Навчання*

Як можна побачити ми також не змогли дістатися вірного результату та побачити як нейронна мережа з кожною наступною спробою збільшує свою винагороду. Один із факторів, який на це впливає це кількість спроб. Тепер спробуємо збільшити кількість спроб до 1000 та подивимося на результат випробування на рисунку 30.

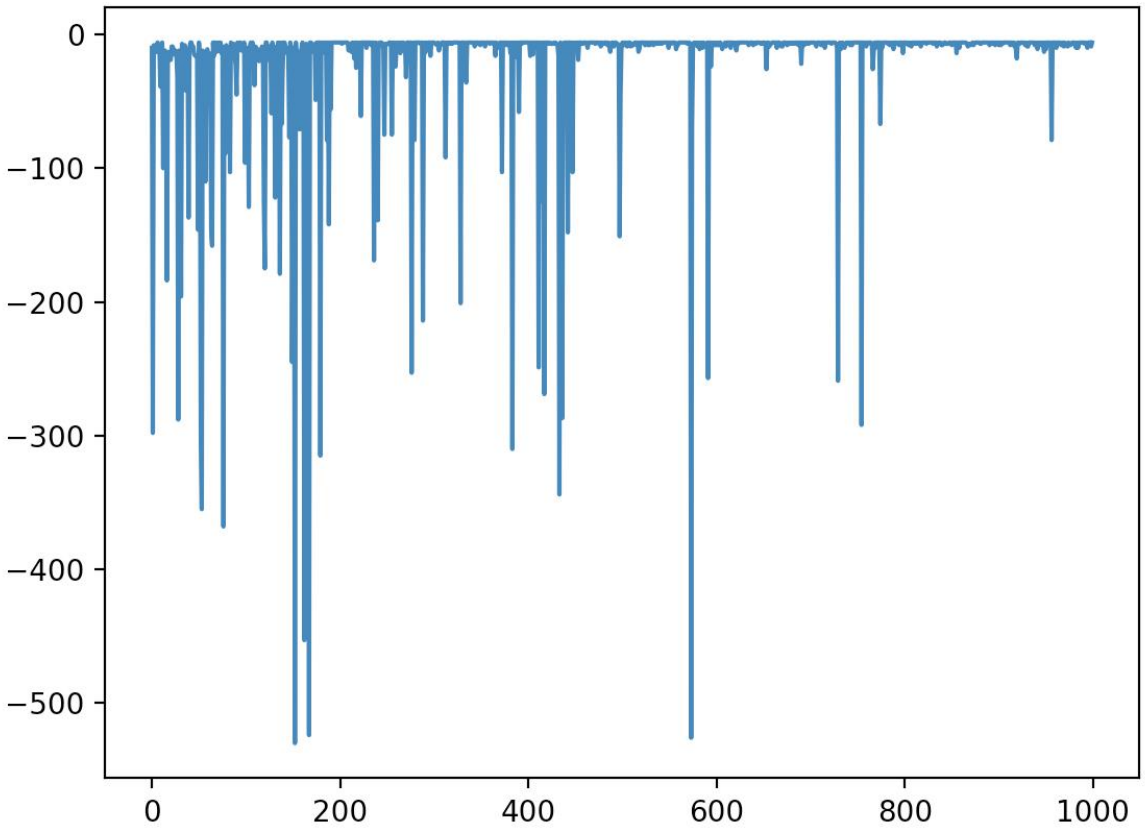

Рис. *30 Навчання*

Тепер можна придивитися більш детальніше на відрізок спроб з 300 до 400ї. Результат на рисунку 31.

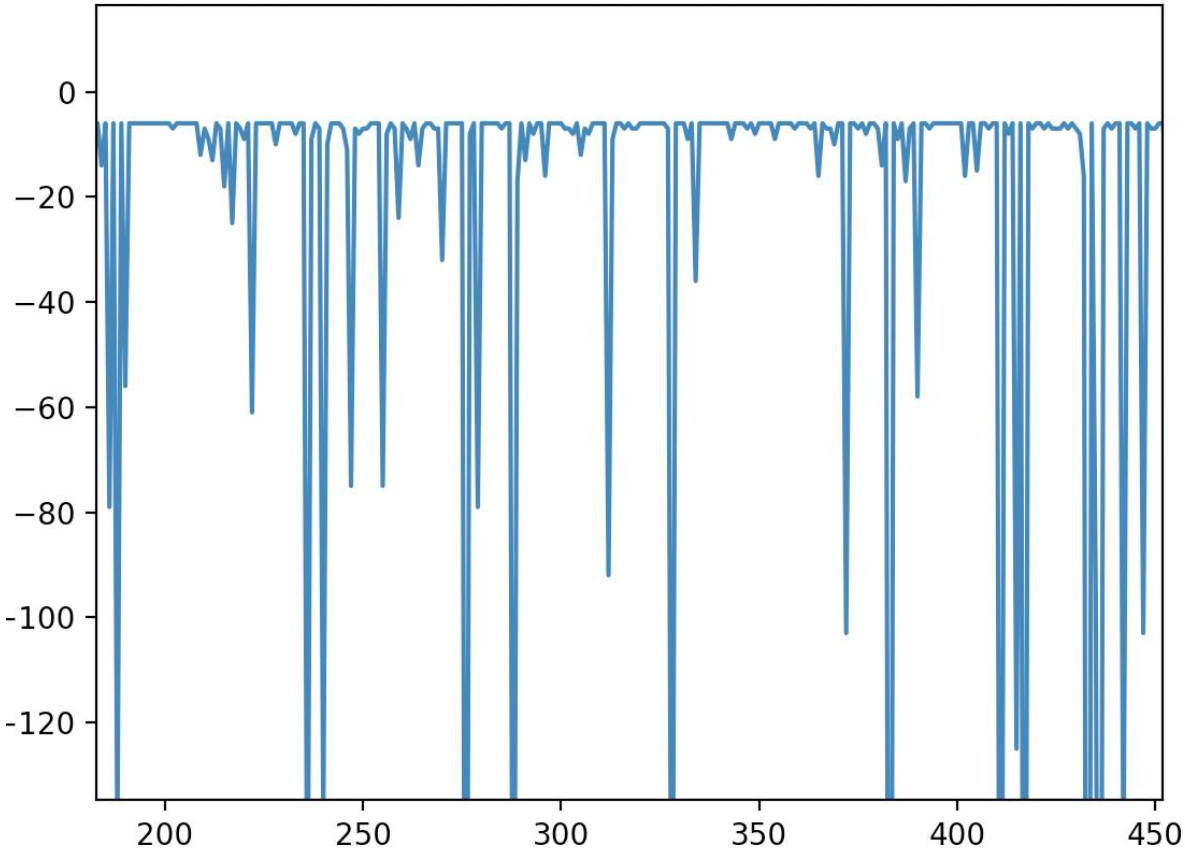

Рис. *31 Навчання*

Як можна побачити на проміжку близько до 350 спроби нейронної мережі вона вже змогла знайти рішення. Щоб перевірити достовірність вищесказаного поставимо “заглушку” у коді, яка після 350-ї спроби змінить значення епсілон до 0 щоб агент тепер спирався лише на Q-таблицю, яку він заповнив за 350 спроб до цього. Результат зображено на рисунку 32.

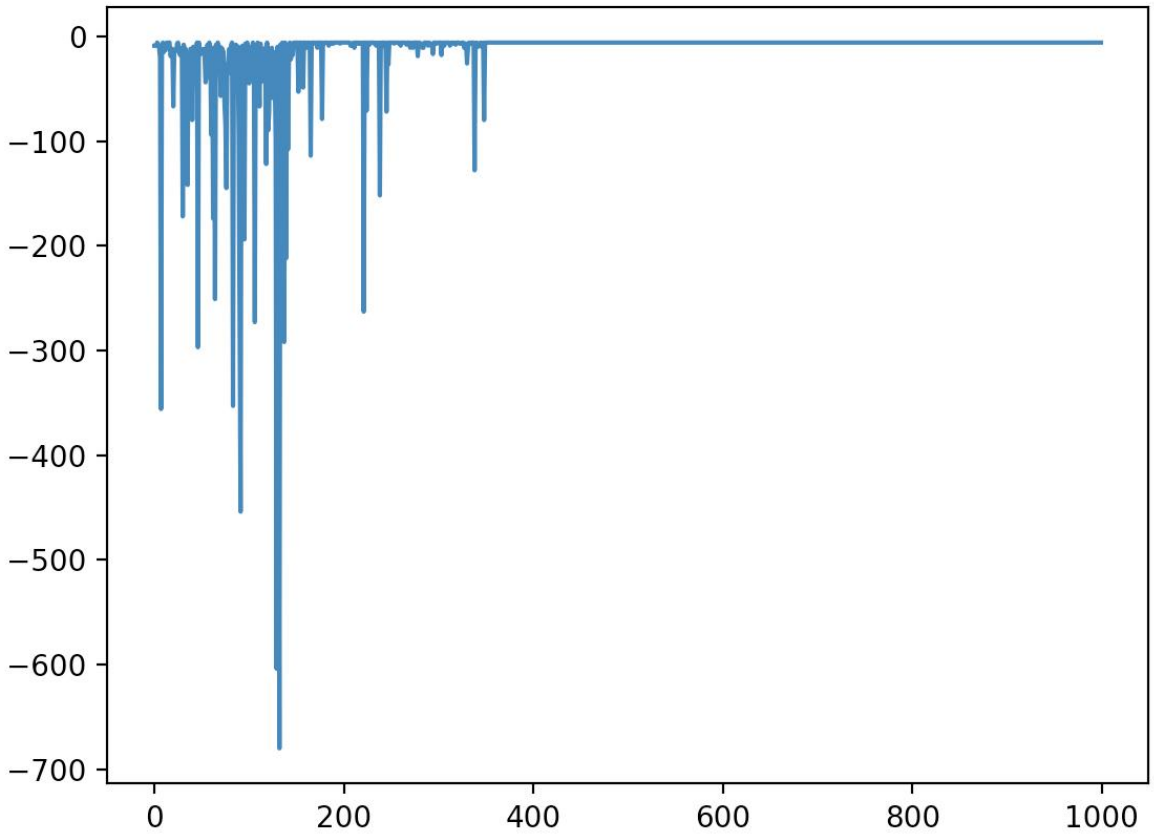

Рис.*32 Навчання*

Як можна побачити агент насправді знайшов рішення та коли ми відкинули можливість випадкового вибору агент набрав максимальну кількість винагороди та пройшов симуляцію ще 600 разів.

ВИСНОВКИ

В рамках магістерської роботи отримані наступні результати:

1. Проведено аналіз сучасних рішень реалізації нейронних мереж.
2. Розглянуто та проаналізовано переваги та недоліки розглянутих інструментів.
3. На основі інструментів TensorFlow та Q-learning була створена нейронна мережа.
4. Розроблено додаток для демонстрації навчання нейронної мережі.
5. Розроблена програма застосована для дослідження та моделювання зв’язків у соціумі. Проведено аналіз результатів

АСПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Руденко С.І., магістрант, Заяц В. І., доцент, к.ф.-м.н — науковий керівник. Дослідження і моделювання зв’язків у соціумі за допомогою нейронних мереж. Збірник наукових праць студентів, аспірантів, докторантів і молодих вчених. Молода наука-2021. Запоріжжя: ЗНУ, 2021. Т.5. С. 97-99.
2. Руденко С.І., магістрант, Заяц В. І., доцент, к.ф.-м.н — науковий керівник. Розробка додатку для керування соціальними групами за допомогою нейронної мережі. Матеріали І Всеукраїнської науково-практичної конференції здобувачів вищої освіти, аспірантів та молодих вчених. Актуальні питання сталого наукового-технічного та соціально-економічного розвитку регіонів України. Запоріжжя : ЗНУ, 2021. С. 344-345
3. Круг П.Г. Нейронные сети и нейрокомпьютеры: Учебное пособие по курсу «Микропроцессоры» / П.Г. Круг – М.: Издательство МЭИ, 2002. – 176 с.
4. Акулов П.В. Решение задач прогнозирования с помощью нейронных сетей / Акулов Павел Владимирович. URL: www. .dgtu.donetsk.ua (дата звернення: 28.06.2021).
5. Оссовский С. Нейронные сети для обработки информации / Станислав Осовский. Пер. с польского И.Д. Рудинского. — М.: Финансы и статистика, 2002. - 344 с.
6. Кальченко Д. Нейронные сети: на пороге будущого / Даниил Кальченко // КомпьютерПресс - 2005. - N1. URL: https://www.compr.ru (дата звернення: 28.06.2021).
7. Півошенко, В. В. Кулик, М. С. Іванов, Ю. Ю. Васюра, А. С. Аналіз та експериментальне дослідження методу безмодельного

навчання з підкріпленням. Вісник Вінницького політехнічного інституту. 2019. 10 с.

8. About TensorFlow. URL: https://www.tensorflow.org/about. (дата звернення: 28.06.2021).
9. Habib, N. Hands-On Q-Learning with Python: Practical Q-learning with OpenAI Gym, Keras, and TensorFlow. 2019. – 212 с.
10. Watkins C. J. C. H., Dayan P. Q-learning //Machine learning. – 1992. – Т. 8. – №. 3-4. – С. 279-292.
11. Машинное обучение Azure. URL: https://azure.microsoft.com/ru-ru/services/machine-learning/ (дата звернення:15.11.2021)
12. What Is Amazon SageMaker?. URL: https://docs.aws.amazon.com/sagemaker/latest/dg/whatis.html (дата звернення: 16.11.2021).
13. Dart overview. URL: https://dart.dev/overview (дата звернення: 08.09.2021).
14. Bloc. URL: https://bloclibrary.dev/#/gettingstarted (дата звернення: 08.09.2021).
15. Flutter. URL: https://flutter.dev (дата звернення: 08.09.2021).
16. Stockfish. URL: https://stockfishchess.org (дата звернення: 28.11.2021).
17. Quick, Draw!. URL: https://quickdraw.withgoogle.com/ (дата звернення: 28.11.2021).
18. Shazam. URL: https://www.shazam.com/ru/company (дата звернення: 28.11.2021).

Декларація

академічної доброчесності

здобувача ступеня вищої освіти ЗНУ

Я, Руденко Семен Ігорович, студент_2_курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти semenrudenko16@gmail.com, — підтверджую, що написана мною кваліфікаційна робота на тему «**Дослідження та моделювання зв'зків у соціумі за допомогою нейронних мереж**» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- заявляю, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;

згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-систем, а також на архівування моєї роботи в базі даних цієї системи.

Дата 30.11.2021 Підпис_____ Руденко Семен Ігорович

(студент)

Дата 30.11.2021 Підпис_____ Заяц Валерій Іванович

(науковий керівник)