МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
СУМСЬКИЙ ДЕРЖАВНИЙ УНІВЕРСИТЕТ
Факультет електроніки та інформаційних технологій

Кафедра комп’ютерних наук

Кваліфікаційна робота магістра

ІНФОРМАЦІЙНА ТЕХНОЛОГІЯ КЕРУВАННЯ НЕЙРОННИХ МЕРЕЖ ДЛЯ НАВЧАННЯ БЕЗПІЛОТНИМ АВТОМОБІЛЕМ

Здобувач освіти гр. ІН.м. – 12ан Юрій ПАХОТА

В. о. завідувача кафедри,
доцент, к.т.н. Ігор ШЕЛЕХОВ

Науковий керівник,
доцент, к.т.н. Сергій ПЕТРОВ

Суми 2022

Сумський державний університет
(назва вузу)

Факультет *ЕлІТ* Кафедра *Комп'ютерних наук*
Спеціальність *«122 - Комп'ютерні науки»*

Затверджую:
В. о. зав. кафедри _____
_____
"_____" _____ 20___р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ МАГІСТРА

Пахоті Юрію Олександровичу
(прізвище, ім'я, по батькові)

1. Тема проекту (роботи) *Інформаційна технологія керування нейронних мереж для навчання безпілотним автомобілем*

затверджую наказом по інституту від "____" _____ 20 ___ р. № _____

2. Термін здачі студентом закінченого проекту (роботи)_____

3. Вхідні данні до проекту (роботи) _____
_____
_____
_____

4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити)
1) Аналітичний огляд методів машинного навчання; 2) Застосування нейронних мереж для лінійної проблеми; 3) Постановка завдання й формування завдань дослідження; 4) Огляд технологій, що використовуються під час розробки додатків на JavaScript; 5) Опис основних положень, технологій і критеріїв, що використовуються алгоритмами нейронної мережі; 6) Розробка інформаційної технології; 7) Аналіз результатів.

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

6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання _____

Керівник _____
(підпис)

Завдання прийняв до виконання _____
(підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів дипломного проекту (роботи)	Термін виконання проекту (роботи)	Примітка
1.	Огляд існуючих рішень застосування нейронних мереж та мов програмування		
2.	Постановка задачі та формування завдань дослідження.		
3.	Опис застосування алгоритмів нейронних мереж та лінійної регресії		
4.	Реалізація функціоналу додатку із використанням мови програмування JavaScript		
5.	Оформлення пояснювальної записки до кваліфікаційної магістерської роботи		

Студент – дипломник _____
(підпис)

Керівник проекту _____
(підпис)

РЕФЕРАТ

Записка: 88 стор., 24 рис., 4 табл., 9 додатків, 29 джерел.

Об'єкт дослідження — Інформаційна технологія керування нейронних мереж для навчання безпілотним автомобілем.

Мета роботи — розробка програмного забезпечення яке зможе приймати деякий модуль або алгоритм машинного навчання з метою використання, тестування і вдосконалення.

Результати — проведений аналіз літератури, методі, та інструментів, які дозволяють застосувати той чи інший алгоритм нейронних мереж для їх використання з метою машинного навчання. Також є можливим випробувати й інші модулі послідовностей для їх тестування та подальшого вдосконалення. Після ознайомлення з існуючими рішеннями, було розроблено програмне забезпечення, яке дає змогу у повному обсязі оглянути всі можливості машинного навчання та використати їх у віртуальному середовищі, в якому можливо випробувати, протестувати та виправити помилки, у разі необхідності. Як приклад, веб-додаток застосовує алгоритм прямого поширення для навчання безпілотного автомобіля на трафіку, тренуючи необхідність виявлення та уникнення колізій. Додаток був реалізований за допомогою мови програмування JavaScript.

МАШИННЕ НАВЧАННЯ, АЛГОРИТМ ПРЯМОГО ПОШИРЕННЯ, ІНФОРМАЦІЙНА СИСТЕМА ДЛЯ ВИПРОБУВАННЯ АЛГОРИТМІВ НЕЙРОННИХ МЕРЕЖ, ЛІНІЙНА РЕГРЕСІЯ, ЛІНІЙНА ІНТЕРПОЛЯЦІЯ, АНАЛІТИЧНА СИСТЕМА, JAVASCRIPT

ЗМІСТ

ВСТУП

Випускна робота присвячена проблемі використання різних алгоритмів для машинного навчання деякої нейронної мережі. Об’єктом дослідження слугує певне віртуальне середовище, яке побудоване на мові програмування JavaScript. Проект відтворює імітацію дорожнього трафіку, в якому, на самому старті веб-додатку, з’являються довільна кількість штучних авто, які запрограмовані доїхати якнайдалі по дорозі, намагаючись уникнути будь-яких зіткнень.

Насправді авто, що генеруються завдяки коду програми, поділяються між собою. Ті, які з’являються на початку роботи інтернет-додатку, - це і є ті, що керуються штучним інтелектом. Також є додаткові об’єкти, що трапляються вздовж всієї дорожньої лінії – фіктивні машини, які слугують як перешкоди, завдяки яким відбувається машинне навчання основних.

Серед всіх авто, які були згенеровані на початку роботи додатку, завжди обирається одна – головна, якій присвоюються зчитувальні сенсори. Вони допомагають з виявленням об’єктів попереду. Інші ж також керуються нейронною мережею та навчаються, але не мають датчиків. Це відбувається наступним чином: якщо відбувається зіткнення деяких двох об’єктів, то та що спричинила аварію видаляється з віртуального середовища та вважається недієздатною. За мить після початку додатку, найбільш просунуте по відношенню до дороги авто обирається головною та має зчитувальні сенсори. Якщо вона була видалена, тобто відбулася колізія між даним об’єктом та будь-яким іншим, то по точно такому ж алгоритму датчики «передаються» іншій машині – тій, що найбільш далеко просунулась по відношенню до стартової точки. І так до останнього дієздатного авто.

Це є досить актуальною та добре поширеною проблемою. Звісно ж додаток цієї дослідницької роботи є лише прототипом та зменшеною і спрощеною версією тієї великої нейронної мережі, що використовує найбільш сучасні алгоритми для рішення різних проблем.

Дослідження даної роботи дозволить автору відтворити зменшену адаптацію того, що можливо застосувати по відношенню до, наприклад, реальної проблеми, яка включає в себе використання справжніх автомобілів на сучасних дорогах та трафіку. Це дасть змогу більше поглибитись в предмет та те, як він працює та як влаштована дана технологія, зрозуміти її. Нейронна мережа, що застосовується в цій дослідницькій роботі, може застосовуватись до різних проблем реальності, які вимагають залучання таких дій, як розпізнавання об’єктів, які, наприклад, містяться на зображенні або в іншому просторі (скажімо, дорожній трафік).

1 ОГЛЯД ІСНУЮЧИХ РІШЕНЬ

Одна із головних цілей даної дослідницької роботи полягає у тому, щоб розробити необхідний веб-додаток, що буде використовувати машинне навчання деякої нейронної мережі. Остання, в свою чергу, буде управляти згенерованими авто на штучний дорожній трасі. Вона спочатку буде намагатися «вчитися» з нуля. Тобто, це буде відбуватись ітераційними циклами, кожний з яких буде отримувати нові та все більш детальні знання про те, як штучна мережа повинна поводитись. За допомогою деяких алгоритмів нейронна мережа зможе здобути саме ті знання, які необхідні для правильного розпізнавання об’єктів навколо себе та завдяки чому уникати колізій зі штучно згенерованими машинами, які відтворюють дорожній трафік.

1.1 Нейронні мережі

Нейронні мережі, також відомі як штучні нейронні мережі (ШНМ) або змодельовані нейронні мережі (ЗНМ), це основа алгоритмів глибокого навчання та є підмножиною машинного навчання [1]. Назва та структура цих технологій натхненні людським мозком, які імітують спосіб, за допомогою якого нейронні сигнали передаються один одному у біологічному середовищі.

Основною одиницею людського мозку є нейрон. Крихітний шматочок мозку, розміром приблизно з рисове зерно, містить понад 10 000 нейронів, кожен з яких утворює в середньому 6 000 з’єднань з іншими нейронами. Завдяки цій величезній біологічній мережі, можливо виконувати ті дії, на які здатні люди [2].

По суті біологічний нейрон оптимізований для отримання необхідної інформації від інших нейронів, її обробки своєрідним та унікальним способом, та надсилання результату до інших клітин. Останнє відбувається за допомогою аксонів. Отримування вхідних даних можливе за допомогою структур, подібних до антен, які називаються дендритами. Вони динамічно посилюються або послаблюються від вхідних з’єднань, в залежності від того,

як часто воно використовується. Таким чином відбувається навчання, і саме сила кожного із з’єднань визначає яким буде внесок вхідних даних у вихідний результат. Для цього зважується міцність вхідної інформації – відповідних з’єднань – і підсумовується в тілі клітини. Потім створюється новий сигнал із цієї суми та надсилається до інших нейронів.

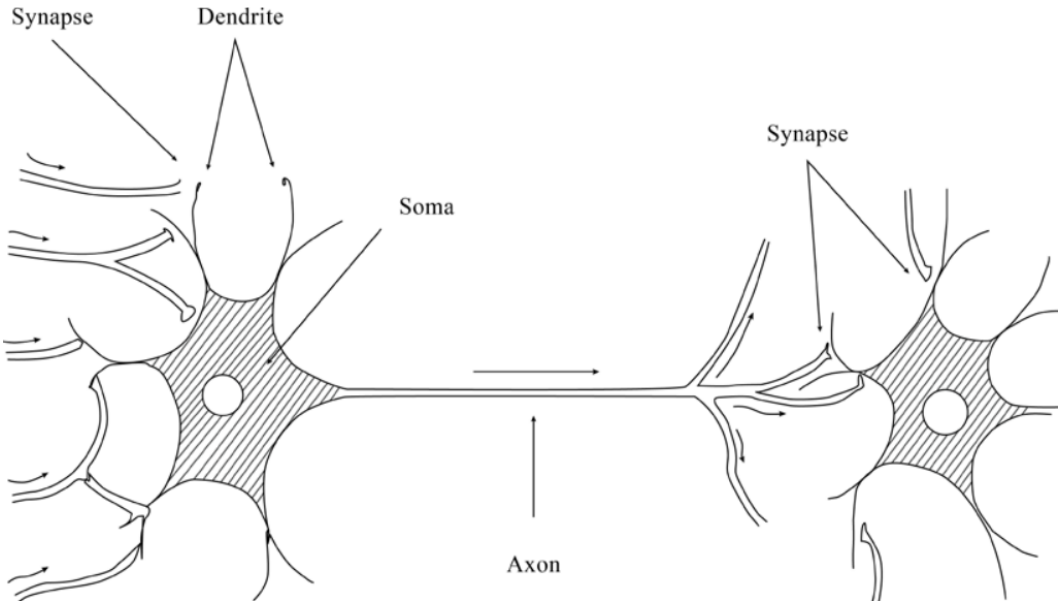

Рисунок 1.1 – нейронні клітини

Мета цієї дослідницької роботи використати цю природну структуру для створення моделі машинного навчання. Нейронні мережі складаються, як не складно здогадатися, з нейронів, що відшаровуються один від одного, для поступового навчання та удосконалення «мозку» штучного інтелекту. Вони можуть містити безліч таких рівнів, але завжди має бути присутній один вхідний та вихідний шари. Від початку і до кінця також можливо, що є один або більше прихованих рівнів. Кожна точка з’єднується з іншою та має відповідну асоційовану вагу та поріг. Головна мета у цій схемі у тому, що якщо результат будь-якого окремого штучного нейрону на будь-якому рівні перевищує вказане порогове значення, то він активується, надсилаючи необхідні дані на наступний шар мережі його зв’язаному вузлу. В іншому випадку точка не робить жодних дій та не передає ніяку вихідну інформацію.

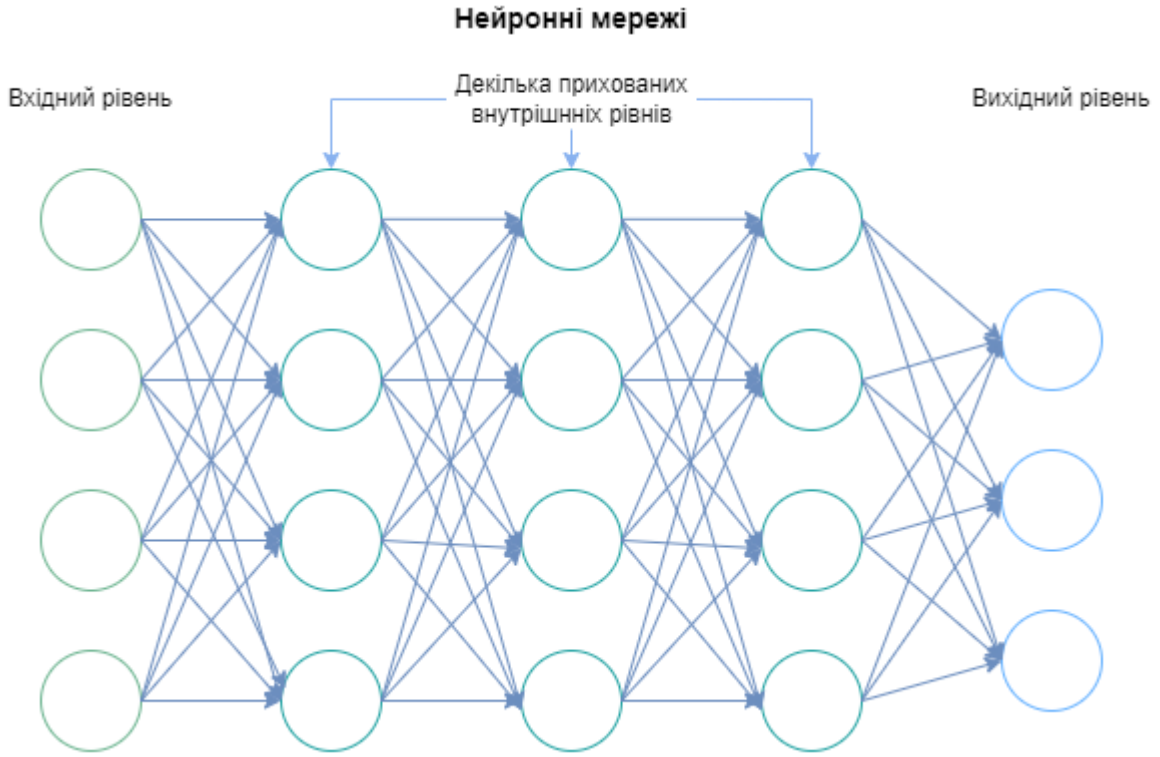

Рисунок 1.2 – приклад роботи нейронних мереж

Нейронні мережі можуть бути класифіковані на різні типи, які використовуються для досить різних цілей. Основні типи даної технології це:

1) Перцептрон
2) Багатошаровий перцептрон
3) Нейронна мережа прямого поширення
4) Згорточна нейронна мережа
5) Мережа радіальних базисних функцій
6) Рекурентна нейронна мережа
7) Довготривала короткочасна пам’ять
8) Модель «послідовність до послідовності»
9) Модульна нейронна мережа

Перший тип цього списку є найстарішою нейронною мережею, що була створена Френком Розенблатом у 1958 році [3]. Вона складається з лише одного нейрону та є найбільш простою формою нейронної мережі:

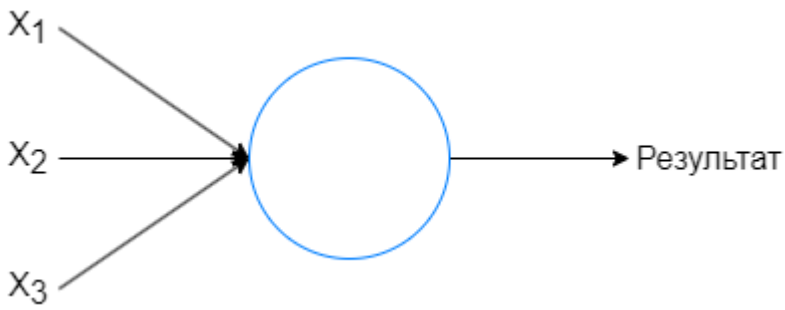

Рисунок 1.3 – Перцептрон

Нейронна мережа типу багатошарового перцептрону (БШП) – це один із підтипів алгоритму прямого поширення. По суті, даний використовує не один нейрон, а декілька, пошарово зв'язаних. Такий підхід додає складності та щільності, завдяки багатьох прихованих шарів між вхідним та вихідним рівнях. Кожен окремий вузол на певній ступені мережі з'єднаний зі всяким вузлом на наступному шарі. Це означає, що даний тип являє собою повністю підключеною між собою нейронною мережею, яку можливо використовувати для глибокого навчання. Вона зазвичай використовує нелінійну функцію, яка ще зветься функцією активації, що і призводить до деякого результату [4]. Кожний вхідний параметр має свій ваговий коефіцієнт. Останні потім додаються та застосовуються в функції, що дає наслідок. Такі штучні мережі застосовуються для більш складних проблем і завдань, які включають в себе нелінійні дані: класифікація або розпізнавання голосу.

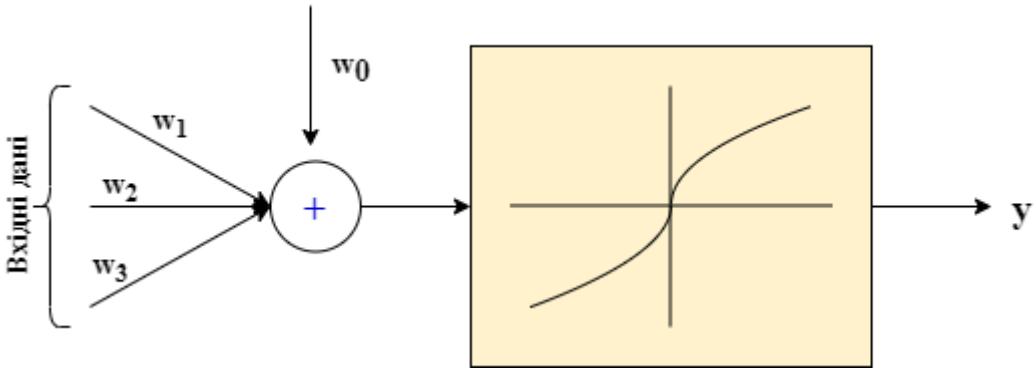

Рисунок 1.4 – принцип багатошарового перцептрону

Нейронні мережі прямого поширення – це основні типи штучних мереж, які в першу чергу розглядаються та на яких зосереджуються. Як було вже згадано вище, вони складаються з початкового вхідного шару, прихованого внутрішнього одного або більше шарів, та результуючого вихідного шару. Ці типи нейронних мереж насправді складаються з сигмовидних нейронів, а не з перцептронів, оскільки більшість реальних проблем є нелінійними. В ці моделі дані зазвичай подаються для навчання, і вони є основою для обробки природної мови, комп'ютерного зору та інших типів штучних мереж [5]. Приклад такої штучної мережи було вже оглянуто на початку цього розділу (рис. 1.2).

Згорткові нейронні мережі схожі на свого попередника зі списку – мережі прямого поширення, але останні зазвичай використовуються для розпізнавання зображень, різних образів або комп'ютерного зору. Цей же тип мереж застосовує принципи лінійної алгебри, зокрема множення матриць, щоб ідентифікувати різні шаблони в необхідному зображені [6].

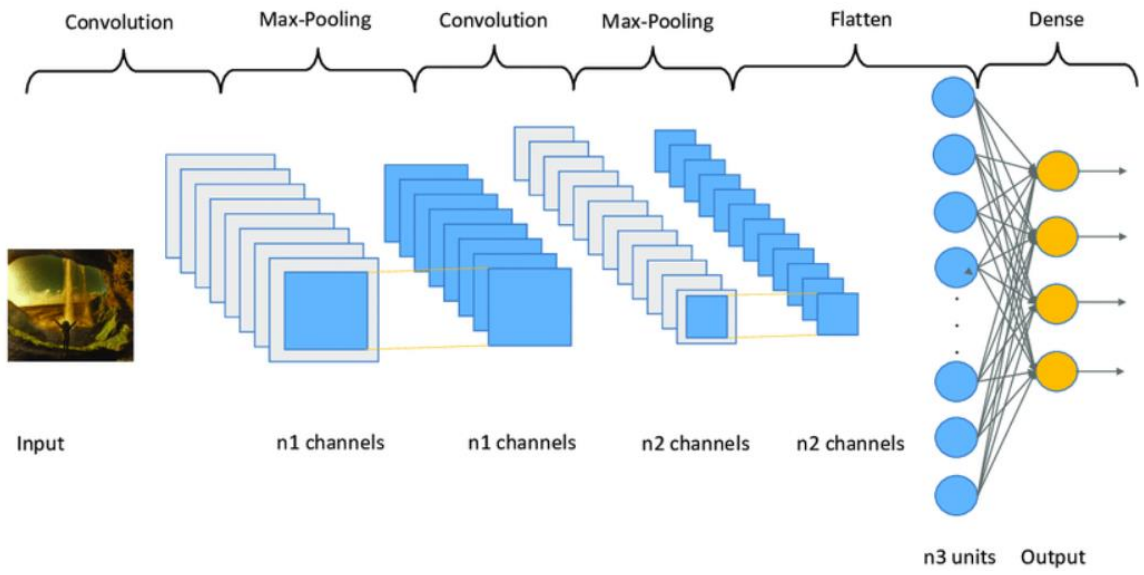

Рисунок 1.5 – принцип роботи згорткової нейронної мережі

Нейронні мережі радіальної базисної функції зазвичай мають вхідний шар, рівень із вузлами радіальної базисної функції з різними параметрами, та вихідну ступінь. Моделі можна використовувати для класифікації, регресії для

часових рядів, і для керування системами. [7] Функції радіального базису обчислюють абсолютне значення між центром і наданою точкою. У випадку застосування нейронної мережі для класифікації, радіальна базисна функція вираховує відстань між вхідними даними та вивченою класифікацією. Якщо водна інформація найближче до певного тегу, вона систематизується як така.

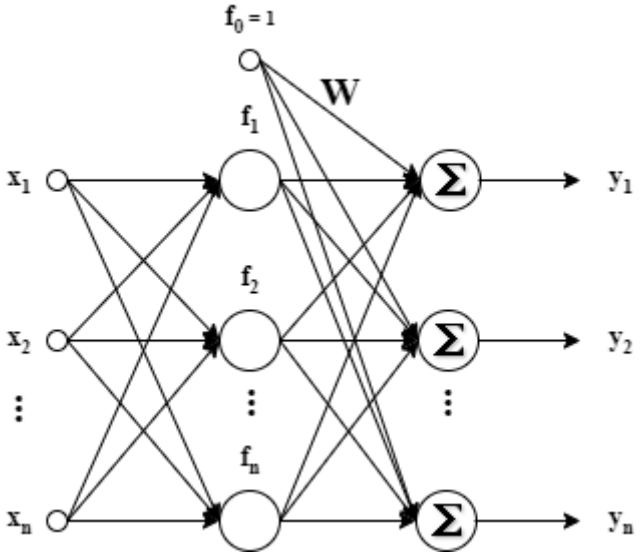

Рисунок 1.6 – архітектура нейронної мережі радіально-базисної функції

Загальне використання нейронних мереж з радіальною базисною функцією включає в себе керування системами, наприклад тими, які контролюють відновлення живлення після відключення або перебою. Штучну мережу можна закодувати таким чином, щоб вона могла розуміти пріоритетний порядок виконання певних дій, щоб запобігти невдачі.

Радіальна базисна функція – це функція, значення якої залежить лише від відстані від початку координат. По суті, вона повинна містити лише дійсні значення. Альтернативні форми радіальних базисних функцій визначаються як відстань від іншої точки, позначеної як C, яка називається центром. Визначення відстані від початку до центру робиться шляхом включення абсолютного значення функції. Вони позначаються як значення без відповідного знаку, який ідентифікує позитивність чи від’ємність.

Рекурентні нейронні мережі ідентифікуються за їх петлями зворотного зв’язку – системи, яка зазвичай означає, що результат попередньої інформації

використовується як вхідні дані для наступної. Дані алгоритми навчання мережі в основному застосовуються разом з даними часових рядів (або дані з міткою часу), що являють собою послідовність точок інформації, проіндексованих у часовому порядку. За допомогою даного процесу як раз і відбувається прогнозування майбутніх результатів – наприклад, передбачення фондового ринку чи пророкування продажів. [7]

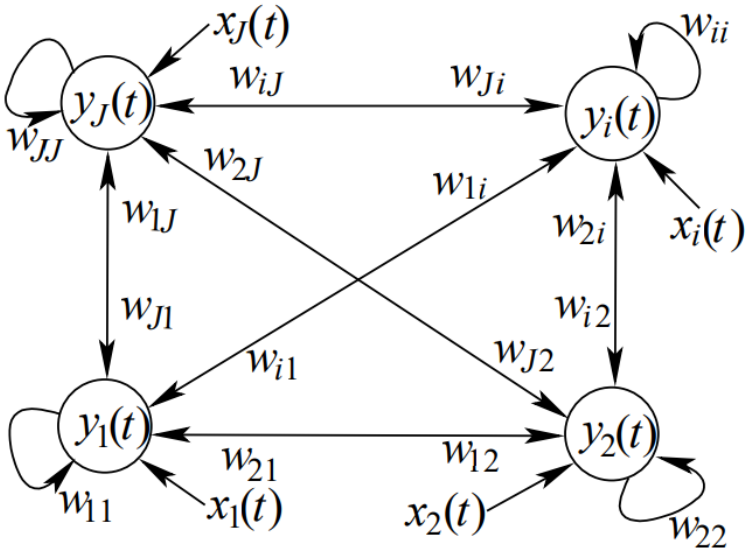

Рисунок 1.7 – архітектура з’єднаної рекурентної нейронної мережі

Модульна нейронна мережа складається з декілька штучних мереж або компонентів, які працюють разом, але незалежно один від одного, для досягнення кінцевого результату. Таким чином, будь-яке складне завдання можливо розбити на більш дрібні компоненти та застосувати всередині даної мережі. [8] Оскільки таке застосування менших елементів відкриє можливість працювати в тандемі, то швидкість обробки буде збільшена, якщо використати штучну мережу до обробки будь-яких даних або обчислювального процесу. Кожен компонент мережі виконує деяке підзавдання, яке в результаті призведе до завершення загального завдання.

Рисунок 1.8 – одна із варіацій модульної нейронної мережі

Для порівняння нейронних мереж нижче наведена таблиця, в якій є коротка інформація про кожну та їх переваги і недоліки.

Таблиця 1.1 – порівняння нейронних мереж

Нейронна мережа	Застосування
Перцептрон	Є найбільш простим варіантом нейронної мережі. Недостатньо для використання та обчислення того, що потрібно в цій роботі.
Багатошаровий перцептрон	Застосовується з нелінійними даними та більш складними задачами. Це не даний випадок, отже такий тип нейронної мережі не підійде.
Прямого поширення	Використовується частіше всього для навчання деяких даних. Має гнучкий та легко зрозумілий алгоритм, що має змогу працювати з будь-якою інформацією: лінійною або нелінійною. Для потреб тренування даних підходить найкраще.

Згорточна нейронна мережа	Застосовує більш складний алгоритм перемноження матриць та підходить краще для проблем із використанням та розпізнаванням зображень.
Радіально-базисної функції	Її зазвичай використовують для класифікації об’єктів та даних, а тому така нейронна мережа не має переваг у цьому дослідженні.
Рекурентна мережа	Як було згадано раніше, в основному застосовується для роботи з даними часових рядів, або інформації з міткою часу, що є деякою послідовністю.
Модульна нейронна мережа	Немає потреби, адже в даній роботі не застосовується декілька модулів, а лише один, з однією метою – навчання.

Після аналізу всіх переваг та недоліків кожного із типів нейронних мереж, для задоволення потреб даної дослідницької роботи, було прийнято рішення обрати нейронну мережу прямого поширення. Саме її буде використано як основний тип штучної мережі, а її алгоритми будуть застосовуватись для машинного навчання штучного інтелекту, що відповідатиме за розпізнавання об’єктів та керування безпілотним авто.

1.2 Мова програмування

Машинне навчання передбачає написання автором деякого коду, що допомагає різним комп’ютерам приймати якесь рішення на основі інструкцій, які випливають із написаного коду, що називаються алгоритмами. Це і є основною концепцією, яка стоїть за всією технологією штучного інтелекту (ШІ). Така сукупність методів з кожним роком все більше і більше набирає

темп зросту, створюючи неймовірні реалізації для вирішення сучасних проблем.

Для того, щоб опанувати всю область машинного навчання та навчитися будувати різні програмні додатки, необхідно вивчити і розуміти одну з мов програмування, які найбільш підходять для цієї технології. Нижче наведений список найбільш переважних розробниками мов кодування, від найбільш популярних до найменш:

1. Python
2. JavaScript
3. R
4. Java
5. C++

Python – це інтерпретована об’єктно-орієнтована мова програмування високого рівня (high-level) з динамічною семантикою [9]. Існує декілька причин чому розробники обирають саме цю технологію, а не іншу мову кодування. По перше, він має велику кількість бібліотек та фреймворків – мова Python «з коробки» постачається з багатьма ресурсами, які економлять час, полегшують та підвищують ефективність розробки додатків. Не потрібно нічого писати, адже люди це вже зробили до цього у вигляді модулів. Ці бібліотеки та фреймворки стають дуже важливими, коли йде мова про застосування машинного та глибокого навчання.

Наступний пункт на захист цієї мови програмування – це лаконічність та читабельність навіть для недосвідчених розробників, що корисно для різних проектів із застосуванням машинного та глибокого навчання [10]. Завдяки простому синтаксису, процес написання коду є досить швидким порівняно з багатьма іншими мовами кодування. Це також дозволяє автору тестувати його роботу та алгоритми без їх впровадження.

Третім пунктом стане масштабна онлайн-підтримка Python спільноти. Оскільки мова програмування є проектом з відкритим кодом, то вона користується відмінною підтримкою багатьох ресурсів і якісної документації.

Також більшість науковців прийняли дану мову кодування для різних проектів, пов’язаних з машинним та глибоким навчанням, тому в спільнотах Python існує багато допомоги.

JavaScript є другою за популярністю мовою програмування, після переможця Python, у сфері штучного інтелекту. Це текстова мова кодування, яка дозволяє створювати не тільки інтерактивні елементи на веб-сторінках, а й цілі веб-додатки та мікросервісні рішення [11]. Даний список не обмежується лише перерахованими сферами застосування, насправді, на JavaScript можна створити будь-що, включаючи розробку додатків для машинного та глибокого навчання [12].

JavaScript хоч і менше застосовується для рішень штучного інтелекту, але він є найбільш популярною мовою програмування в цілому. Це означає, що JavaScript є більш відомим у світі, а тому підтримка у розробці будь-якого веб-додатку може бути навіть ще сильніше, ніж надає Python.

Окрім цього, JavaScript має вбудовану безпеку. Більшість програм машинного навчання покладаються на клієнт-серверну архітектуру. Користувачі повинні надсилати свої дані там, де працюють алгоритми штучного навчання. Однак у багатьох випадках краще виконувати результат машинного навчання на пристрої абонента з міркувань безпеки. Саме тут «вступає в гру» JavaScript, який підтримується всіма сучасними мобільними та настільними браузерами та додатками. Це означає, що розроблене рішення машинного навчання гарантовано працює на більшості пристроїв.

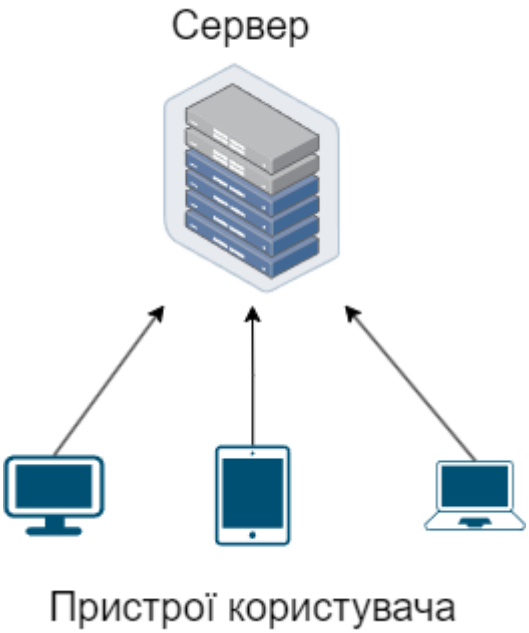

Рисунок 1.9 – клієнт-серверна архітектура

Завдяки JavaScript можливо дуже легко інтегрувати машинне навчання та його реалізацію у будь-який веб- або мобільний додаток. Інші мови програмування все ще знаходяться на попередніх стадіях, а тому це є вагомим плюсом, якщо розроблювати програми саме під ці платформи.

R – це вільна мова програмування, яка містить обширний каталог статистичних і графічних методів [13] та включає в себе алгоритми машинного навчання [14], а також багато інших. Аналіз даних за допомогою даної мови кодування відбувається в декілька кроків: програмування, перетворення, виявлення, моделювання та передача результатів.

R був створений для таких специфічних типів, як статистичний аналіз та візуалізація даних. Основна перевага даної мови кодування перед іншими – це те, що весь функціонал необхідний для проведення таких обчислювальних операцій, є основою, яка одразу вбудована в R. Для більшості основної роботи, які можливо виконати за допомогою цієї мови програмування, ніякі бібліотеки та фреймворки не потрібні.

Java – також одна із найбільш популярних мов програмування, яка є об’єктно-орієнтованою мовою кодування загального призначення, що

базується на класах та розроблена для меншої кількості реалізаційних залежностей [15]. Це обчислювальна платформа для розробки додатків. Її також можливо застосовувати для машинного навчання [16].

Java використовує віртуальну машину (Java Virtual Machine або JVM) для компіляції коду в машинний, і тому її вважають однією з найкращих платформ для штучного навчання та науки про дані. Вона дозволяє розробнику писати код, який є ідентичним на багатьох установах.

Java також має чудову масштабованість, що є великим аспектом мови програмування на сьогоднішній день, який розробники враховують перед початком кожного проекту. Це відкриває можливості для створення великих або складних програм штучного інтелекту та машинного навчання.

Багато інших широко використовуваних мов програмування для машинного навчання не є найшвидшими варіантами, а тому Java чудово підходить під ці потреби. Вона також має велику кількість різних бібліотек та інструментів для роботи зі штучним інтелектом.

C++ - це ще одна із багатьох мов програмування, що підтримує об’єктно-орієнтовану парадигму, високого рівня з швидшим часом виконання порівняно з більшістю інших інструментів для кодування [17]. Це все завдяки тому, що він ближче до машинної мови, яку сприймають комп’ютери. Як і попередня мова, C++ є досить швидкою технологією, а також має багато бібліотек та інструментів для роботи зі штучним навчанням [18]. З іншої сторони, всі ці привабливі особливості потребують деякої ціни. Це включає в себе не таку велику спільноту, як, наприклад, у перших двох пунктах, а також необхідність слідкувати за різними проблемами мови програмування вручну. Останнє під собою має на увазі спостерігання за різними витоками пам’яті, що пов’язані з комп’ютерним обладнанням.

Після огляду всіх мов програмування, треба підсумувати всю обговорену інформацію у вигляді таблиці порівняння.

Таблиця 1.2 – порівняння мов програмування

Мова програмування	Застосування
Python	Має багато ресурсів та модулів щодо машинного навчання, а також велику спільноту у цій сфері, але потрібні додаткові зусилля та час на візуалізацію всього процесу.
JavaScript	Одна із найпопулярніших мов у світі. Також має велику спільноту у всіх сферах, а також багато ресурсів та модулів. Головним плюсом є те, що чудово поєднується з браузером та можливістю візуалізації будь-яких даних, написаних на цій мові.
R	Застосовується більше для статистики, аналізу, та праці з даними (очистка та побудування на основі деякої інформації). Завдяки таким особливостям вона більше підходить для таких сфер, як генетика, біоінформатика, медицина, та епідеміологія.
Java	Java є масштабованою, а тому більше підходить проблем корпоративного рівня. До того ж, вона не має таку велику кількість різних бібліотек та модулів для праці з машинним навчанням, як той же Python або JavaScript. Це є досить рішучим мінусом, не кажучи вже про те, що немає інтеграції з браузером не застосовуючи веб-сервер.
C++	Так, C++ досить швидка мова програмування порівняно з іншими, але немає потреби в її використанні. Більшість бібліотек та модулів, які застосовуються в інших технологіях, написані також на C++. Тому додаткового підвищення продуктивності це не надає. І, знову ж таки, виникає потреба в реалізації з’єднання частини з візуалізацією до самого коду, написаного на C++.

Проаналізувавши всі плюси і мінуси всіх п’ятьох мов програмування, було прийнято рішення використовувати інструмент для кодування JavaScript. Він не сильно відрізняється по простоті, лаконічності та читабельності Python, а тому добре пригодиться як для новачків, так і для вже досвідчених розробників. Спільнота JavaScript насправді є такою ж великою, як і в Python, тому що вона є однією з найбільш популярних в світі. Це, в свою чергу, означає добру підтримку та велику кількість бібліотек та інструментів, готових до використання. Як було згадано раніше, JavaScript має міцнішу безпеку, оскільки вся робота виконується на пристрої користувача даної системи. Оскільки на меті стоїть візуалізувати всю роботу та відтворити у вигляді веб-сторінки, то JavaScript тут виграє перед Python. Перша мова програмування вбудована в браузер, і легко може виконати такі потреби. Алгоритм написаний без використання зовнішніх бібліотек та модулів, а тому в інших інструментах для написання коду зовсім немає додаткової необхідності. Також, проект не має на меті застосовувати велику кількість даних для створення об’ємної нейронної мережі, а тому можливість вбудованих в мову програмування алгоритмів та швидкість не грають масивної ролі в даному випадку.

1.3 Постановка задачі

Головною метод даної дослідницької роботи є створення інформаційної технології – аналітичного веб-додатку – який дозволить застосувати деякий алгоритм машинного навчання для тренування певної нейронної мережі.

Розроблена програма повинна вміти:

- відтворювати каркаси для відображення шосе та дій самої штучної мережі;
- генерувати деяку кількість вхідних авто для їх подальшого навчання;
- створювати імітацію трафіку на дорозі;
- реалізувати алгоритм нейронної мережі, яка буде управляти машинами;

- програмно обирати «головну» машину та віддавати контроль над нею нейронній мережі;
- втілити недопустимість колізій між об’єктами та видалення пошкодженого авто в такому разі;
- запрограмувати передачу нейронній мережі контролю над авто до другої при зіткненні;
- реалізувати навчання даних нейронної мережі з кожною новою ітерацією;
- створити можливість збереження та видалення інформації про штучну мережу на поточній послідовності.

Для досягнення поставленої мети потрібно вирішити наступні задачі:

a) провести огляд існуючих рішень;
b) обрати засоби та методи дослідження для вирішення задачі;
c) реалізувати програмне середовище.

2 МЕТОДИ ДОСЛІДЖЕННЯ

Історія нейронних мереж набагато довша, ніж більшість людей думають про неї. Ідея «машини, яка мислить» ще прослідковувалась у часи стародавніх греків, але у даній роботі розглядається лише та ключова інформація, що має безпосередній зв’язок до подій, які призвели до еволюції мислення навколо нейронних мереж, популярність яких з роками то спадала, то зменшувалася.

Одна із перших таких подій, яка складає ланцюжок поступового розвитку даної технології, є робота Уоррена Мак-Каллока і Уолтера Піттса під назвою «Логічне числення ідей, властивих нервовій діяльності», що була випущена у 1943 році [19]. Ці дослідження мали на меті зрозуміти, як людський мозок у свій біологічний шлях може виробляти складні моделі через зв’язані клітини мозку або нейрони. У процесі даної роботи, однією із головних ідей, що зародилася у авторів, було порівняння нейронів із двійковим порогом до булевої логіки. Остання являє собою один із типів алгебри, який зосереджений навколо двох слів, відомих як булеві оператори: «так» та «ні». Вони ще можуть позначатися як «1» (один) та «0» (нуль), що визначаються як «істина» та «хиба» відповідно, та відомі як істинні значення або істинні змінні.

Наступна робота, що досліджувалась та дала значний прорив в області нейронних мереж – це праця Френка Розенблата, якому приписують розробку та імплементацію перцептрона, задокументовану в його досліджені під назвою «Перцептрон: ймовірнісна модель для зберігання та організації інформації в мозку», випущеному у 1958 році [3]. Він зміг просунути роботу МакКалоха та Пітта на крок далі, додаючи ваговий коефіцієнт та вже відоме рівняння ваги. Застосовуючи комп’ютер від компанії IBM, Розенблат зміг навчити та змусити техніку розрізняти картки, що були позначені ліворуч, від тих, що знаходилися праворуч.

Згадуючи метод зворотного поширення, першою людиною, що зробила внесок у сторону нейронних мереж та використала його у цій технології був

Пол Вербос, який описав свій досвід та застосування у власній докторській дисертації у 1974 році [20].

У 1989 відбулося перше використання обмежень у методі зворотного поширення, а також його інтеграція в архітектуру нейронної мережі, що було описано у статті від Янна ЛеКун [21]. Це дало змогу застосовувати технологію для навчання різних алгоритмів, і тому, як наслідок, за допомогою цього дослідження було успішно використано нейронну мережу для розпізнавання рукописних цифр поштового індексу, наданих Поштовою службою США.

2.1 Алгоритми нейронних мереж

Штучні нейронні мережі застосовують та покладаються на навчання даних, що спочатку надходять до вхідного рівня, та підвищують свою точність завдяки тренуванню, наближаючись до потрібної вихідної інформації з кожним наступним шаром. Через такий підхід, проходячи через всі рівні мережі, на останньому ступені дані знаходяться на піку стадії навчання і вважаються найточнішими та засвоєними. Це все є потужним інструментом в різних областях, таких як штучний інтелект та інформатика в цілому, оскільки машинне навчання націлене, в основному, на поправність та відповідність. Це дозволяє автоматично класифікувати дані з неймовірною швидкістю, що означає, що такі технології, як розпізнавання голосу, зображення, або тексту, є реальністю та займає лічені години, хвилини, або навіть секунди. У порівнянні з людськими можливостями, навіть якщо працюють професіонали свого діла, це просто неймовірний прорив, що економить велику кількість сил, ресурсів та часу. Одне з чудових прикладів роботи цієї технології – це пошуковий алгоритм, який використовує, наприклад, компанія Google у своїх програмних додатках.

Кожен вузол, який є основою для будування всіх рівнів у нейронній мережі, являє собою власну модель лінійної регресії, що сама по собі складається з вхідних даних, вагових коефіцієнтів, зміщення (або порогу) та вихідних даних.

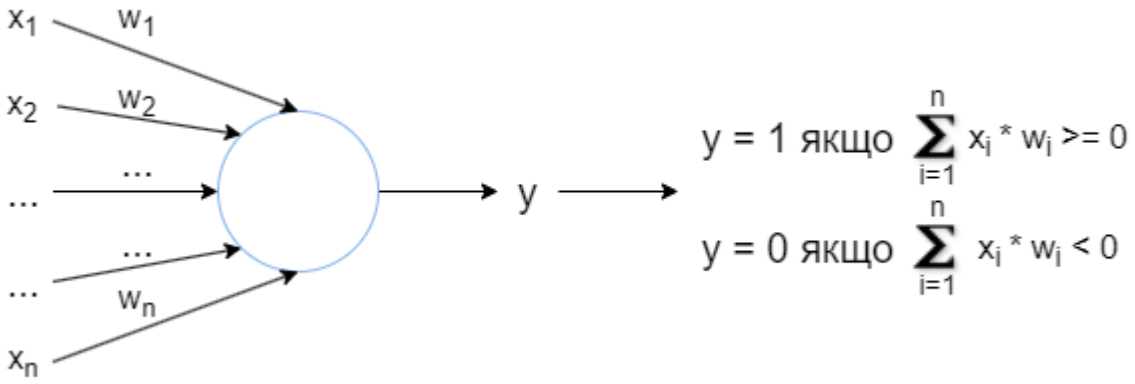

Рисунок 2.1 – лінійна регресія

Після того, як вхідний рівень був визначений, призначається його відповідна вага. Це значення допомагає охарактеризувати важливість будь-якої змінної, що була задана. Причому більші вагові коефіцієнти вносять більший внесок у кінцевий результат порівняно з іншими вступними параметрами. У результаті всі вхідні дані множаться на їх відповідні вагові значення, а після – підсумовуються. Далі всі вихідні дані мають пройти через функцію «активації», яка визначає подальші вихідні результати. Якщо кінцева результативна інформація перевищує задане зміщення (або поріг), то спрацьовує (або активується) певний вузол, що потім передає дані на наступний рівень нейронної мережі. Як наслідок, це призводить до того, що вихідні показники одного вузла стають вхідними параметрами для наступного рівня штучної мережі. Цей весь процес передачі інформації від одного рівня до іншого повністю визначає нейронну мережу як мережу прямого зв’язку. Формула такої взаємодії виглядає наступним чином:

$$\sum_{i=1}^{m} w_i\, x_i + bias = w_1 x_1 + w_2 x_2 + \cdots + w_i x_i + bias$$

$$y = f(x) = \begin{cases} 1, \text{якщо } \sum w_1 x_1 + b \geq 0 \\ 0, \text{якщо } \sum w_1 x_1 + b < 0 \end{cases}$$

Для того, щоб зрозуміти як може виглядати один окремий вузол нейронної мережі, необхідно розібратись за допомогою двійкових значень (0 або 1). Як приклад, береться більш відчутний випадок – чи варто займатися будь-яким видом спорту (так – 1, ні – 0). Таке рішення (йти у спорт чи не йти) і є прогнозованим результатом всього процесу застосування лінійної регресії. Вихідний параметр ($\hat{y}$) означає передбачене значення у (залежної зміни) у будь-якому рівнянні. Воно ще також вважається середнім значенням змінної відповіді. Припустимо, що на прийняття рішення впливають декілька факторів:

a) Чи підходить людина по стану здоров’я?
b) Чи має бажаючий достатню кількість часу?
c) Чи є можливість записатися на потрібну секцію спорту?

Приймемо такі відповіді:

a) $x_1 = 1$, стан здоров'я підходить.
b) $x_2 = 0$, немає достатньої кількості часу.
c) $x_3 = 1$, так як є можливість.

Після визначення вхідних даних, необхідно призначити деякі вагові коефіцієнти, щоб визначити важливість параметрів. Більші ваги будуть означати, що певні зміни мають більшу важливість для рішення та більш впливають на кінцевий результат.

a) $w_1 = 4$, оскільки стан здоров’я відповідає необхідному.
b) $w_2 = 1$, оскільки часу зовсім недостатньо для відвідування занять.
c) $w_3 = 3$, оскільки можливість є, але є ймовірність пропустити заняття.

І нарешті, припускається порогове значення, наприклад, в 3, що буде означати відповідність зсуву як -3. Маючи на увазі всі вищезгадані вхідні дані, є можливість почати додавати оглянуту інформацію у формулу та отримати бажаний кінцевий результат:

$$\hat{y} = 1 * 4 + 0 * 1 + 1 * 3 - 3 = 4$$

Якщо застосувати у цьому процесі функцію активації, отримується визначення, що вихідний результат такого вузла буде 1 (так), оскільки 4 більше

ніж 0. У даному випадку ймовірність записатися на будь-який вид спорту – вкрай висока (відповідь – так), але є деякий нюанс: якщо налаштувати вагові коефіцієнти або поріг, то кінцевий результат буде відрізнятися від поданої моделі. Це все підкреслює та наголошує на тому, що нейронна мережа може приймати дедалі складніші рішення в міру проходження за рівнями, та в залежності від інформації, яку вона отримує на попередніх шарах та висновках.

У наведеному вище прикладі було використано так званий перцептрон – алгоритм навчання – щоб проілюструвати як він працює та яку роль грає математика в даному процесі. Штучні мережі застосовують інші нейрони – сигмоподібні, що відрізняються значеннями від 0 до 1. Оскільки нейронні мережі поводять себе подібно до дерев рішень, каскадуючи дані з одного вузла до іншого, наявність значення x від 0 до 1 добре так зменшить вплив будь-якої змінної на вихідний результат деякого взятого вузла, що пов'язаний з нею, а згодом – як наслідок – і на всю мережу та її продукт.

Кажучи вже про більш практичні випадки, використання таких нейронних мереж, як, наприклад, розпізнавання та класифікація зображень або тексту, має на меті застосовувати для цих процесів контрольоване навчання, або мічені набори даних, що активно беруть участь у навчанні алгоритму. Для того, щоб оцінити точність технології, використовується функція витрат. Її ще зазвичай називають середньоквадратичним відхиленням (СКВ) [22].

$$\text{СКВ} = \frac{1}{2m}\sum_{i=1}^{m}(\hat{y} - y)^2, \text{ де:}$$

- і представляє індекс вибірки.
- $\hat{y}$ – прогнозований результат.
- y – фактичне значення.
- m – розмір вибірки.

Зрештою, мета всієї цієї процедури полягає в тому, щоб мінімізувати дану функцію витрат. Робиться це для того, щоб забезпечити коректність відповідності для будь-якого заданого спостереження. Коли модель коригує

свої ваги та зміщення (або пороги), вона використовує вищезгадану функцію вартості та підкріплююче навчання для того, щоб досягти певну точку конвергенції або локального мінімуму. Такий процес, у якому алгоритм коригує свої вагові коефіцієнти, здійснюється за допомогою градієнтного спуску, що дозволяє моделі здійснити визначення напрямку для зменшення похибок (або мінімізувати функцію витрат) [23]. З кожним наступним навчальним прикладом параметри моделі коригуються для подальшого поступового зближення до мінімуму.

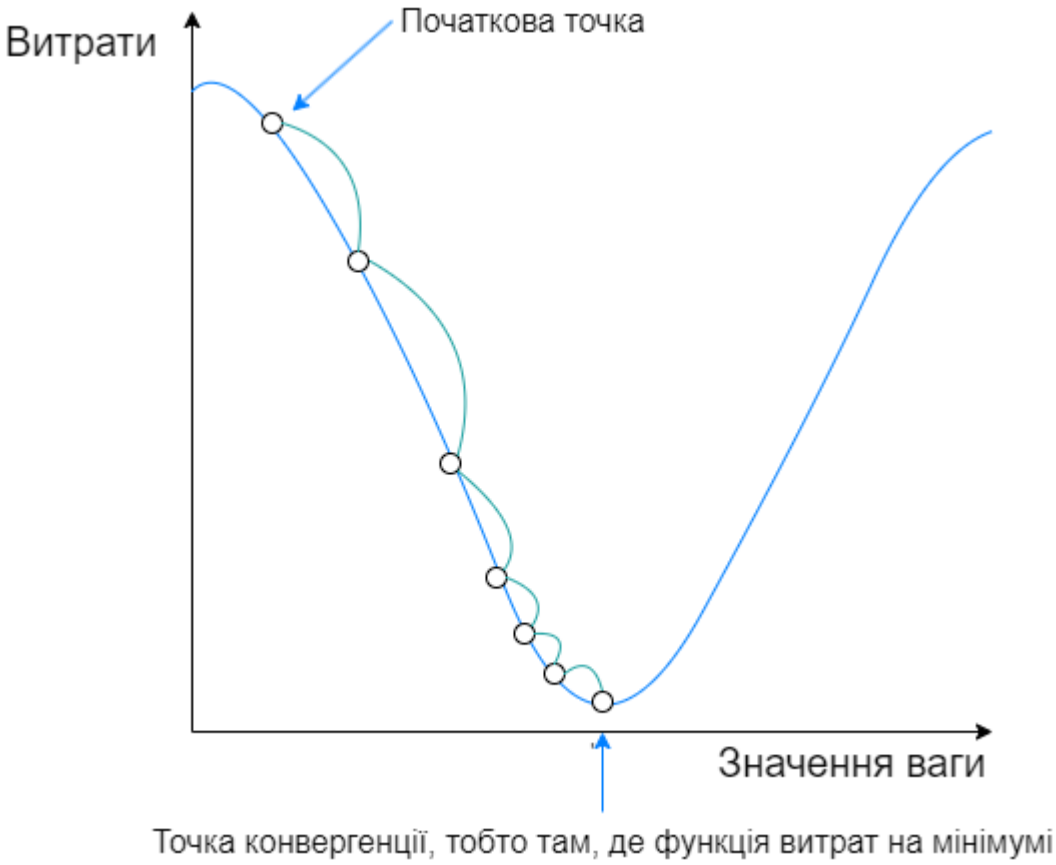

Рисунок 2.2 – функція витрат

2.2 Лінійна регресія

Лінійна регресія, або її аналіз, застосовується для прогнозування деякої змінної на основі значення іншої змінної. Це використання параметру для передбачення стану іншої змінної називається незалежним значенням. [24]

Застосування такого підходу дає можливість оцінити коефіцієнти лінійного рівняння, що включає одну або декілька незалежних змінних, які найкраще передбачають значення первинної змінної. Лінійна регресія відповідає прямій лінії або поверхні, що максимально мінімізує розбіжності між прогнозованими та фактичними вихідними даними. Існує багато створених простих калькуляторів лінійної регресії, які використовують метод найменших квадратів, що допомагають виявити найкращої лінії для певного набору парних даних. Це дає змогу оцінити значення залежної зміни (x) та незалежної зміни (y).

Чому це так важливо? Моделі лінійної регресії є відносно нескладними та дають змогу забезпечити просту для інтерпретації математичну формулу, яка може генерувати подальші прогнозування. Цей підхід може застосовуватися до різних математичних сфер, а також сфер бізнесу та академічних досліджень. На практиці, лінійна регресія використовується у всьому: починаючи від біологічних, поведінкових та екологічних науках, та завершуючи соціальними і бізнес аспектами життя. Її застосування вже стало перевіреним шляхом для наукового та надійного прогнозування вихідних даних та майбутнього. Оскільки цей підхід є давно встановленою статистичною процедурою, властивості використання лінійної регресії вже добре зрозумілі та нескладні для швидкого розуміння та навчання.

На практиці лінійна регресія може застосовуватись для, наприклад, збирання маси даних, які потім використовуються для кращого керування реальністю – замість покладання на досвід, інтуїцію та власну статистику. Цей підхід дозволяє брати великі обсяги необроблених даних та перетворювати їх на корисну та практичну інформацію.

Для успішного вираховування прогнозування та максимально наближених до реальності даних, використовуються так звані ключові припущення ефективної лінійної регресії.

Таблиця 2.1 – ключові припущення

Ключове припущення	Дії
Для кожної змінної	Необхідно розглянути припустиму кількість дійсних випадків, їх середнє значення та стандартне відхилення
Для кожної моделі	Потрібно перевірити коефіцієнти регресії, кореляційну матрицю, частину та часткові кореляції, кілька R, R2, відрегульований R2, зміни в R2, стандартну похибку оцінки, таблицю дисперсійного аналізу, прогнозовані значення та залишки. Крім того, необхідно враховувати 95-відсоткові довірчі інтервали для кожного коефіцієнта регресії, дисперсійно-коваріаційної матриці, коефіцієнта інфляції дисперсії, толерантності, тесту Дарбіна-Ватсона, показників відстані (Махаланобіса, Кука та значення кредитного плеча), DfBeta, DfFit, інтервалів прогнозування та діагностичної інформації для кожного випадку.
Для графіків	Роздивитися діаграми розсіювання, часткові графіки, гістограми та графіки нормальної ймовірності.
Для даних	Залежні та незалежні змінні мають бути кількісними. Категоріальні змінні, такі як, наприклад, релігія, основні галузі навчання або регіони проживання, перекодовуються в бінарні (фіктивні) змінні або інші типи контрастних змінних.

Інші припущення	Для кожного значення незалежної змінної на будь-якому рівні розподіл залежної змінної має бути нормальним. Дисперсія розподілу останньої повинна бути постійною для всіх значень першої. Зв’язок між залежною змінною та всіма її незалежними змінними повинен бути лінійним, а всі спостереження будуть незалежними

Для того, щоб застосувати лінійну регресію до будь-якої проблеми, необхідно спочатку переконатися, що вхідні дані взагалі можливо проаналізувати за допомогою цієї процедури. Важливо, щоб початкова інформація проходила через певні потрібні припущення. Для виконання останніх, можливо застосувати такі дії:

1) Змінні слід вимірювати на постійному рівні. Прикладами такого вимірювання є, наприклад, час, вага, продажі або результати тестів. Для того, щоб швидко з’ясувати, чи існує лінійний зв’язок між двома змінними, застосовується діаграма розсіювання
2) Будь-які спостереження мають бути незалежними один від одного, а це означає, що не повинно бути ніякої залежності.
3) Дані не мусять мати значних викидів.
4) Перевірка на гомоскедастичність – статистична концепція, згідно якої дисперсії вздовж найбільш відповідної лінії лінійної регресії залишаються подібними на всій лінії.
5) Похибки найкращої лінії регресії мають відповідати нормальному розподілу.

2.3 Нейронні мережі проти глибокого навчання

Ці дві технології, як правило, взаємозамінні в розмові, але зазвичай таке трактування може заплутати. Як наслідок, варто зазначити, що слово «глибина» у глибокому навчанні означає лише кількість рівнів певної нейронної мережі. Остання ж може включати в себе декілька шарів, зокрема більше трьох, які обов'язково включатимуть в себе початкову та кінцеву. Така штучна мережа зазвичай вважається алгоритмом глибокого навчання. Застосовуючи тренування нейронів за допомогою штучної мережі, якщо вона має два або три рівні, то її приймають за просто базову нейронну мережу.

Глибоке навчання – це одна із багатьох частин машинного навчання. Основна різниця між ними полягає в тому, як кожна із цих технологій застосовує алгоритми, як вони навчаються, та скільки даних використовує кожен тип алгоритму. Глибоке навчання ще відрізняється тим, що робить більшу частину процесу вилучення функцій автоматизованим, тим самим зменшуючи необхідність безпосереднього ручного втручання людини. Ця технологія також дозволяє взаємодіяти з великими наборами даних, що підводить до перспективи масштабування, заслуживши назву «машинне навчання з можливістю масштабування». Така спроможність буде особливо корисною при праці та дослідженні неструктурованих даних.

Класичне, або «неглибоке», машинне навчання ж більше залежить від безпосереднього втручання людини для тренування. Певна кількість експертів, що стоїть за навчанням, визначають деяку ієрархію особливостей, щоб зрозуміти основні відмінності між вхідними даними. Зазвичай для вивчення потребуються більш структуровані дані. Крім того, можливо використовувати різні мітки для позначення вхідної інформації та спростування процесу тренування за допомогою навчанню під наглядом.

Існують відомі контрольовані навчання, коли «глибоке» машинне навчання може застосовувати вищезгадані позначені набори даних, але воно не обов'язково потребує такої інформації з мітками. Технологія може приймати неструктуровані дані в необробленому вигляді (наприклад, текст,

зображення), і автоматично визначати такий набір ознак, за якими показники відрізняють один від одного.

Спостерігаючи над такими закономірностями у даних, модель глибокого навчання може належним чином кластеризувати вхідну інформацію. Це означає, що є правдоподібним застосувати групування схожих, наприклад, зображень або тексту, у відповідні категорії на основі деяких відмінностей, що були визначені. Таким чином, модель глибокого навчання вимагатиме більше точок даних для більш підвищеної точності результату, тоді як класична технологія покладається на меншу кількість інформації, враховуючи базову структуру цих даних. Архітектура глибокого тренування в основному застосовується для більш складних випадків використання, як-от розпізнавання аудіо та мови, машинного перекладу, фільтрації соціальних мереж, виявлення шахрайства, віртуальні помічники тощо.

3 РЕАЛІЗАЦІЯ ІНФОРМАЦІЙНОЇ ТЕХНОЛОГІЇ

3.1 Формування вхідних даних

Для реалізації задуманого плану та відтворення дослідницької роботи у вигляді програми, необхідно визначити декілька вхідних даних. Оскільки це віртуальне середовище, в якій буде шосе, на якому є деякі машини, що імітують трафік та постійно рухаються вперед.

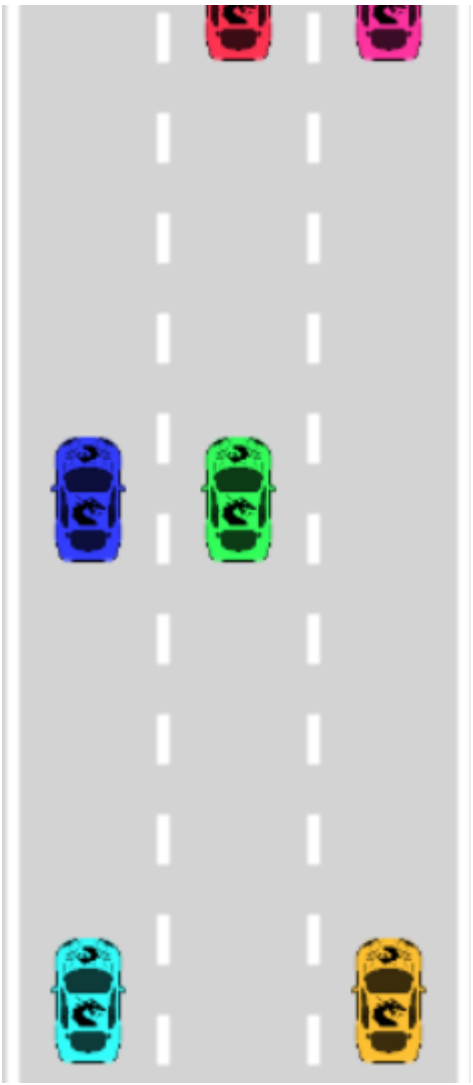

Рисунок 3.1 – шосе зі згенерованими авто

Для машинного навчання нейронної мережі, необхідно на початку роботи додатку згенерувати декілька, або багато, авто, які будуть намагатись тренуватись «на ходу» та уникати зіткнення з вищезгаданим трафіком, який буде попереду. Кількість створених початкових машин залежить від вхідної

даної N, яка, наприклад, у даному випадку складає $N = 100$. Тобто на старті програми згенерується 100 авто, які незалежно один від одного будуть намагатись проїхати по шосе, уникаючи будь-яких зіткнень з іншими об’єктами, та опираючись на свій досвід з попередніх ітерацій та випробувань.

Рисунок 3.2 – генерування N об'єктів на початку додатку

Також для розуміння та відтворення результатів праці машинного навчання нейронної мережі, необхідно створити деяке інтерактивне «поле» для того, щоб спостерігати за усім процесом тренування дослідницької мережі. Це штучне «полотно» буде таким собі дисплеєм, що відтворюватиме поведінку нейронної мережі та показуватиме, що саме відбувається в той чи інший момент часу, а також які дії вона робить.

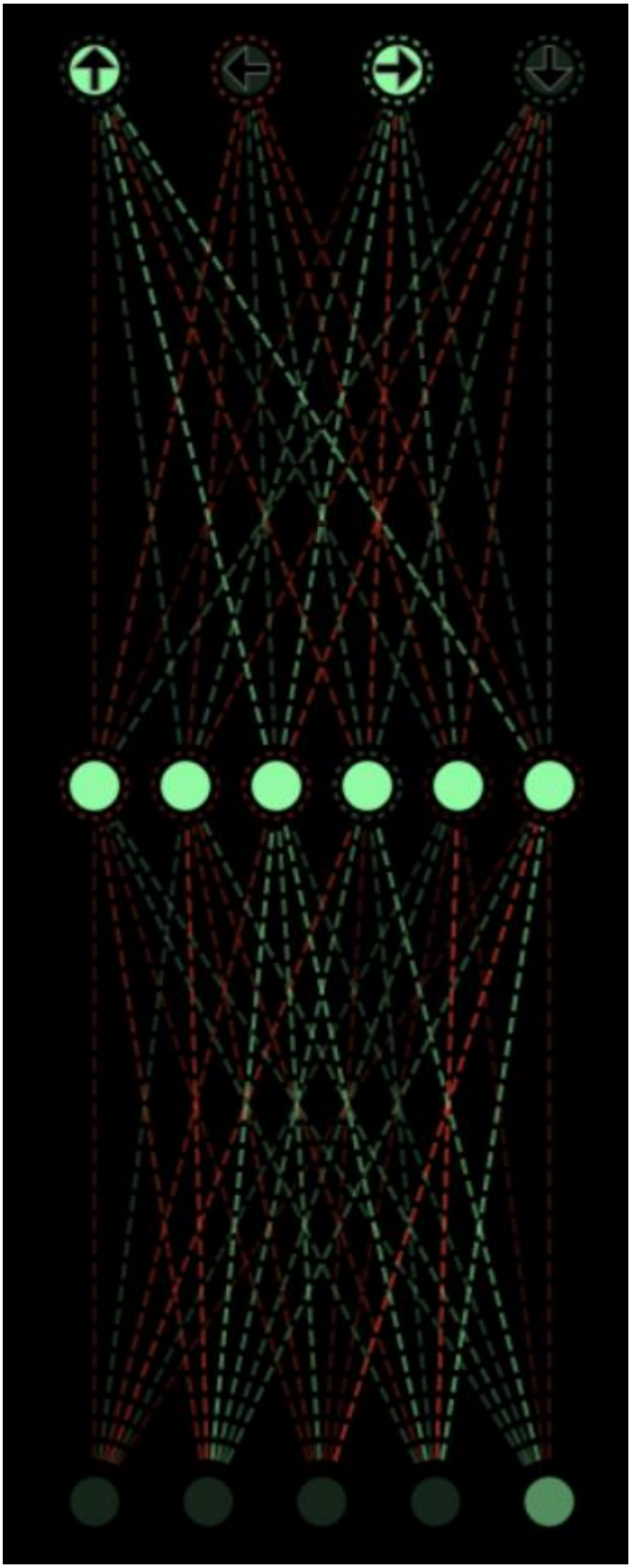

Рисунок 3.3 – відтворення поведінки нейронної мережі

Для створення штучної мережі типу прямого поширення, необхідно задати вхідні дані у вигляді кількості потрібних нейронів, що потім будуть складати рівні самої нейронної мережі, її основу. Це масив, що складається із цифр, кожна з яких є чисельністю штучних нейронів на відповідному шарі використаної мережі. Вхідна інформація, тобто масив цифр, потім застосовується для ітерації по ньому та формування кожного із рівнів нейронної мережі, що в свою чергу має вхідні дані у вигляді кількості поточних нейронів, а також тих, що будуть в наступному шарі.

Таким чином, вхідні дані для реалізації даної дослідницької роботи складаються з декілька пунктів, які йдуть у порядку їх виконання:

1) Відтворення шосе.
2) Імітування дорожнього трафіку.
3) Генерація початкових авто для навчання.
4) Створення та під'єднання нейронної мережі.
5) Відтворення поведінки мережі.

3.2 Програмна реалізація

На самому початку програми створюється два «полотна», які відповідають за те, щоб «відмалювати» місце для відображення самого шосе з трафіком, а також поведінки самої нейронної мережі, щоб було зручніше та зрозуміліше слідкувати за її діями. За це відповідає вбудований в мову програмування JavaScript рідний клас під назвою *HTMLCanvasElement*:

```
/**
 * @type {HTMLCanvasElement}
 */
const carCanvas = document.getElementById('car-canvas');
carCanvas.width = 200;

/**
 * @type {HTMLCanvasElement}
 */
const networkCanvas = document.getElementById('network-canvas');
networkCanvas.width = 300;

const carCtx = carCanvas.getContext('2d');
const networkCtx = networkCanvas.getContext('2d');
```

Код 3.1 – рендеринг полотнів під шосе та нейронної мережі

Префікс HTML, що стоїть перед назвою класу, означає застосування браузером мови тегів (tags), які використовуються для створення веб-сторінок – саме те, що потрібно для даної дослідницької роботи. Акронім розшифровується як мова гіпертекстової розмітки (Hyper Text Markup Language). Вона є стандартом для вироблення веб-сторінок та веб-сайтів. Технологія дозволяє створювати та структурувати розділи, абзаци, та посилання, за допомогою вбудованих елементів (по суті, будівельних «блоків» сторінки), таких як теги та атрибути. [25]

При створені полотна та класу HTMLCanvasElement, дана функція приймає так званий ідентифікатор, на якому буде розміщуватись саме цей об’єкт, для подальшої роботи з ним. Для того, щоб правильно та відповідно його відобразити відносно інших елементів веб-сторінки, необхідно додати та «накласти» стилі на нього. Для таких цілей на допомогу приходять каскадні таблиці стилів, або CSS (Cascading Style Sheets), - проста мова дизайну, яка призначена для того, щоб зробити деякий веб-сайт виглядити більш презентабельно. Технологія доповнює HTML та керує зовнішнім виглядом веб-сторінки, роблячи можливим керування кольором тексту, стилем шрифтів, інтервалами між абзацами, розміром і розташуванням стовпців та зображень, і в цілому дизайном сайту [26].

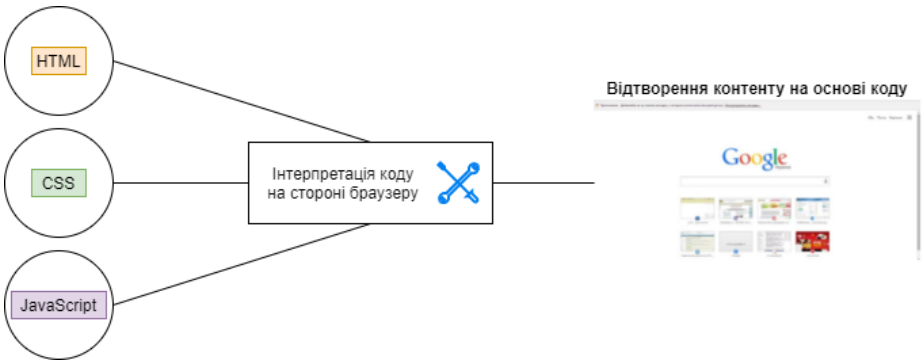

Рисунок 3.4 – відтворення роботи веб-сайту

Після того, як полотно було створено, на нього необхідно «помістити» об’єкти, і в цілому реалізувати всю необхідну логіку, що притаманна трафіку:

центр по осі абсцис Ox, її ширину, кількість дорожніх розміток, ліву та праву границі по осі x, верхню та нижню границі, чотири точки по куткам самого прямокутника, а також бордюри дороги.

```
const road = new Road(carCanvas.width / 2, carCanvas.width * 0.9);

const infiniteValue = 1000000;

export class Road {
  /**
   * @param {number} x A value to center the road around
   * @param {number} width A width value of the road
   * @param {number} linesCount The road lines count (default is 3)
   */
  constructor(x, width, linesCount = 3) {
    this.x = x;
    this.width = width;
    this.linesCount = linesCount;

    this.left = x - width / 2;
    this.right = x + width / 2;
    this.top = -infiniteValue;
    this.bottom = infiniteValue;

    const topLeft = {
      x: this.left,
      y: this.top,
    };
    const topRight = {
      x: this.right,
      y: this.top,
    };
    const bottomLeft = {
      x: this.left,
      y: this.bottom,
    };
    const bottomRight = {
      x: this.right,
      y: this.bottom,
    };
    this.borders = [
      [topLeft, bottomLeft],
      [topRight, bottomRight],
    ];
  }
}
```

Код 3.2 – ініціалізація дороги

Всі вищезгадані функції та рівняння зберігаються в такому ж вигляді і в коді програми. За це відповідає технологія JSON, що розшифровується як «JavaScript Object Notation». Це досить популярний відкритий стандартний формат для обміну даних, зрозумілий людині. Він незалежний від мови, але

вбудований у JavaScript. Механізм дозволяє зберігати різні типи даних та підтримує майже всі види мов, фреймворків, і бібліотек. [27]

Рисунок 3.5 – приклад даних у форматі JSON

Наступним кроком після створення дороги буде генерація початкових автівок, які керуються нейронною мережею в подальшому:

```
const cars = generateCars(road, 'AI', 100);

/**
 * Generates and returns a number of cars.
 * @param {Road} road A road to build cars on.
 * @param {'DUMMY' | 'AI'} type The type of cars.
 * @param {number} n A number to create.
 * @return {Car[]} The created cars.
 */
export function generateCars(road, type, n = 1) {
  const cars = [];

  for (let i = 0; i < n; i++) {
    cars.push(new Car(road.getLineCenter(1), 100, 30, 50, type));
  }

  return cars;
}
```

Код 3.3 – генерація початкових авто

В даному випадку створюється сто екземплярів, якими штучка мережа намагатиметься керувати по черзі, та подолати гру.

Появлення автівок відбувається на одній із полос дороги, в даному випадку це вибір від 1 до 3, тому що у цьому разі шосе має лише три ділянки.

```
/**
 * Get the center of a road line.
 * @param {number} lineNum
 * @return {number} x
 */
getLineCenter(lineNum = 1) {
  const lineWidth = this.width / this.linesCount;
  return this.left + lineWidth / 2
    + Math.min(lineNum, this.linesCount - 1) * lineWidth;
}
```

Код 3.4 – отримання потрібної дорожньої полоси

Дана функція повертає якусь цифру, що означає місце на полотні по осі абсцис Ox, що потрібна для генерації авто. Окрім цього параметру, остання також приймає координату по осі ординат Oy, ширину та висоту машини, її тип контролю (нейронна мережа або манекен для відтворення трафіку), максимальна швидкість пересування, а також колір для того, щоб розрізняти між різними типами. Клас, який відповідає за генерацію авто, також має такі властивості, як постійна швидкість пересування, значення прискорення, тертя, кут напряму, її місткість, елементи керування, та, якщо тип – штучний інтелект, зчитувальні сенсори і «мозок», управління яким на себе бере штучна мережа.

```
const imgSrc = './assets/car.png';
const ctxDestination = 'destination-atop';

export class Car {
  /**
   * @param {number} x x
   * @param {number} y y
   * @param {number} width The width of a car.
   * @param {number} height The height of a car.
   * @param {'DUMMY' | 'AI'} type The type of car: for making traffic or for
training.
   * @param {number} maxSpeed The maximum speed of a car. The default value
is 3...
   * @param {string} color The color of a car. The default is blue.
   */
  constructor(x, y, width, height, type, maxSpeed = 3, color = 'lightblue') {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.color = color;

    this.speed = 0;
```

```
    this.maxSpeed = maxSpeed;
    this.acceleration = 0.2;
    this.friction = 0.05;
    this.angle = 0;

    this.isDestroyed = false;

    if (type === 'AI') {
      this.sensor = new Sensor(this);
      this.brain = new NeuralNetwork(
        [this.sensor.raysCount, 6, 4],
      );
    }
    this.controls = new Controls(type);

    this.#loadCarImage();
  }

  #loadCarImage() {
    this.image = new Image();
    this.image.src = imgSrc;

    this.carCanvas = document.createElement('canvas');
    this.carCanvas.width = this.width;
    this.carCanvas.height = this.height;

    const canvasCtx = this.carCanvas.getContext('2d');

    this.image.onload = () => {
      canvasCtx.fillStyle = this.color;
      canvasCtx.rect(0, 0, this.width, this.height);
      canvasCtx.fill();

      canvasCtx.globalCompositeOperation = ctxDestination;
      canvasCtx.drawImage(this.image, 0, 0, this.width, this.height);
    };
  }
}
```

Код 3.5 – ініціалізація авто

Як вже було згадано раніше, нейронна мережа повинна постійно навчатись та удосконалюватись в кожному поколінні. Це головний фактор, без якого не було б можливим проводити поліпшення «головного мозку». Для цього було створено спеціально додаткові кнопки збоку шосе, щоб спостерігач мав змогу зберегти саме ту поведінку нейронної мережі, яка, на його думку, здалася йому найкраще та найбільш вигідна для поставленої мети – доїхати до кінця. Перша із кнопок виконує саме останній функціонал – збереження порогового значення та ваги всякого нейрону на кожному рівні. Завдяки цьому, всі згенеровані авто будуть намагатися відтворити вже натреновану поведінку нейронної мережі, а також удосконалити її ще далі, на значення t.

Друга кнопка відміняє та видаляє збережену інформацію про дії штучної мережі та змушує її «почати спочатку».

За цей функціонал відповідають HTML-теги під назвою button, а схороняють дані у спеціальне сховище браузеру, яке має назву localStorage (або локальне сховище).

```
<div class="vertical-buttons">
  <button onclick="save()">Save💾</button>
  <button onclick="discard()">Remove🗑</button>
</div>

/**
 * Store the best brain to the local storage.
 * @param {{
 * levels: {inputs: number[], outputs: number[], biases: number[], weights:
number[]}[]
 * }} brain
 */
export function saveBrainToStorage(brain) {
  saveToLocalStorageItem(brainLabel, brain);
}

export function removeBrainFromStorage() {
  removeItemFromLocalStorage(brainLabel);
}

/**
 * Save an item to the local storage.
 * @param {string} key
 * @param {string} value
 */
export function saveToLocalStorageItem(key, value) {
  localStorage.setItem(key, JSON.stringify(value));
}

/**
 * Remove an item from the local storage.
 * @param {string} key
 */
export function removeItemFromLocalStorage(key) {
  localStorage.removeItem(key);
}

window.save = saveBrainToStorage;
window.discard = removeBrainFromStorage;
```

Код 3.6 – збереження та діставання нейронної мережі

Локальне сховище має властивість зберігати інформацію навіть при перезавантаженні вікна додатку, або навість самого веб-серверу, а також не має терміну придатності. Видалення відбувається лише вручну (або за

допомогою функціоналу додатку), або через налаштування самого браузеру [29].

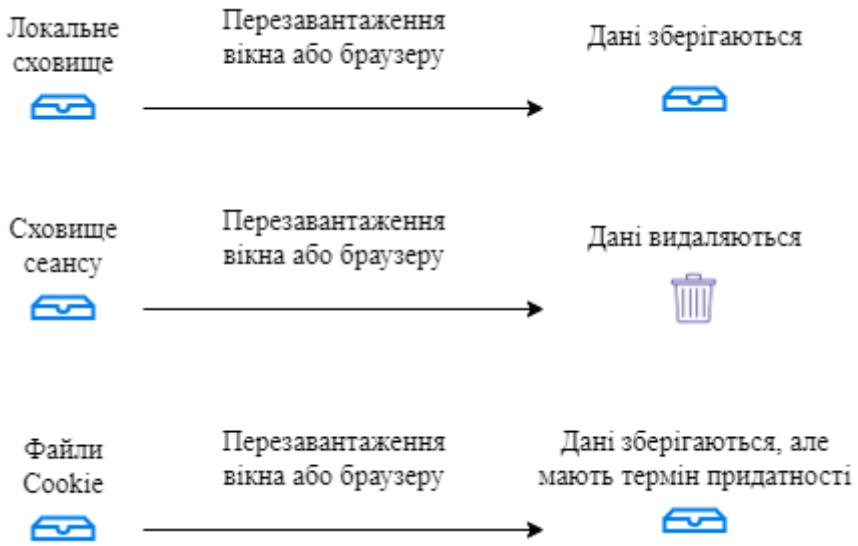

Рисунок 3.6 – порівняння сховищ браузеру

Якщо ж локальне сховище не містить ніякої збереженої інформації з попередніх ітерацій, вона самостійно створюється «з нуля», приймаючи необхідну кількість нейронів для створення на кожному із рівнів. Останні також генеруються під час цього процесу, як окремий JavaScript клас. При цьому для всякого штучного нейрону генеруються випадкові порогові значення та вага, в діапазоні від -1 до 1.

```
export class NeuralNetwork {
  /**
   * A Neural Network to train on cars.
   * @param {number[]} neurons An array of neuron amounts for each level.
   */
  constructor(neurons) {
    this.layers = [];

    for (let i = 0; i < neurons.length - 1; i++) {
      this.layers.push(new NetworkLayer(neurons[i], neurons[i + 1]));
    }
  }

  /**
   * Use the Feed Forward algorithm to train the neural network.
   * @param {number[]} inputs
   * @param {NeuralNetwork} network
   */
  static feedForward(inputs, network) {
    let outputs = NetworkLayer.feedForward(inputs, network.layers[0]);

    for (let i = 1; i < network.layers.length; i++) {
      outputs = NetworkLayer.feedForward(outputs, network.layers[i]);
```

```
    }
    return outputs;
  }

  /**
   * Train the neural network further, by some amount (%).
   * @param {NeuralNetwork} network
   */
  static train(network, amount = 1) {
    network.layers.forEach((layer) => {
      for (let i = 0; i < layer.biases.length; i++) {
        layer.biases[i] = lerp(
          layer.biases[i],
          Math.random() * 2 - 1,
          amount,
        );
      }
      for (let i = 0; i < layer.weights.length; i++) {
        for (let j = 0; j < layer.weights[i].length; j++) {
          layer.weights[i][j] = lerp(
            layer.weights[i][j],
            Math.random() * 2 - 1,
            amount,
          );
        }
      }
    });
  }
}
```

Код 3.7 – генерація нейронної мережі

Окрім самої нейронної мережі потрібно також створити її рівні: по кожному відповідному класу на кожен шар.

```
export class NetworkLayer {
  /**
   * A layer of a Neural Network.
   * @param {number} inputNeuronsCount
   * @param {number} outputNeuronsCount
   */
  constructor(inputNeuronsCount, outputNeuronsCount) {
    this.inputs = new Array(inputNeuronsCount);
    this.outputs = new Array(outputNeuronsCount);
    this.biases = new Array(outputNeuronsCount);

    this.weights = [];
    for (let i = 0; i < inputNeuronsCount; i++) {
      this.weights[i] = new Array(outputNeuronsCount);
    }

    this.#randomize();
  }

  #randomize() {
    for (let i = 0; i < this.inputs.length; i++) {
      for (let j = 0; j < this.outputs.length; j++) {
        this.weights[i][j] = Math.random() * 2 - 1;
      }
    }
    for (let i = 0; i < this.biases.length; i++) {
```

```
      this.biases[i] = Math.random() * 2 - 1;
    }
  }

  /**
   * Use the Feed Forward algorithm to train the neural network.
   * @param {number[]} inputs
   * @param {NetworkLayer} layer
   * @return {number[]} outputs
   */
  static feedForward(inputs, layer) {
    for (let i = 0; i < layer.inputs.length; i++) {
      layer.inputs[i] = inputs[i];
    }

    for (let i = 0; i < layer.outputs.length; i++) {
      let sum = 0;

      for (let j = 0; j < layer.inputs.length; j++) {
        sum += layer.inputs[j] * layer.weights[j][i];
      }
      if (sum > layer.biases[i]) {
        layer.outputs[i] = 1;
      } else {
        layer.outputs[i] = 0;
      }
    }
    return layer.outputs;
  }
}
```

Код 3.8 – ініціалізація класу рівня нейронної мережі

При кожному оновлені кадру та машин, виконується вищезгаданий алгоритм прямого поширення нейронної мережі, таким чином удосконалюючи штучну систему, що призводить до покращеного контролю над авто. Саме тут вступає в гру розрахунок суми ваги кожного рівня нейронної мережі і подальше порівняння із пороговим значенням. Висновком цих обчислень стає необхідність машини повернути в ту чи іншу сторону.

```
const offsets = this.sensor.readings.map(
  (reading) => (reading ? 1 - reading.offset : 0),
);
const [forward, left, right, backward] = NeuralNetwork.feedForward(offsets,
this.brain);

if (this.brain) {
  this.controls.forward = forward;
  this.controls.left = left;
  this.controls.right = right;
  this.controls.backward = backward;
}
```

Код 3.9 – оновлення контролерів авто

Покадрове оновлення також провокує відсвіжування зчитувальних сенсорів авто, які застосовують вищезгадану лінійну інтерполяцію. Це, в свою чергу, кожного разу змушує систему відливати промені, разом з виявленням дистанції до найближчого об’єкту.

```
/**
 * A function to update the sensors of a car.
 * @param {[{x: number, y: number}, {x: number, y: number}][]}
roadBordersCoords
 * @param {Car[]} traffic
 */
update(roadBordersCoords, traffic) {
  this.#createRays();

  this.readings = [];

  for (let i = 0; i < this.rays.length; i++) {
    this.readings.push(
      Sensor.#getReading(
        this.rays[i],
        roadBordersCoords,
        traffic,
      ),
    );
  }
}

#createRays() {
  this.rays = [];

  for (let i = 0; i < this.raysCount; i++) {
    const rayAngle = lerp(
      this.raysSpreadWidth / 2,
      -this.raysSpreadWidth / 2,
      this.raysCount === 1 ? 0.5 : i / (this.raysCount - 1),
    ) + this.car.angle;

    const start = {
      x: this.car.x,
      y: this.car.y,
    };
    const end = {
      x: this.car.x - Math.sin(rayAngle) * this.raysLength,
      y: this.car.y - Math.cos(rayAngle) * this.raysLength,
    };

    this.rays.push([start, end]);
  }
}

/**
 *
 * @param {[{x: number, y: number}, {x: number, y: number}]} ray
 * @param {{ x: number, y: number }[][]} roadBorders
 * @param {Car[]} traffic
 */
static #getReading(ray, roadBorders, traffic) {
  const intersections = [];

  for (let i = 0; i < roadBorders.length; i++) {
```

```
    const touch = getIntersection(
      ray[0],
      ray[1],
      roadBorders[i][0],
      roadBorders[i][1],
    );
    if (touch) {
      intersections.push(touch);
    }
  }

  for (let i = 0; i < traffic.length; i++) {
    const carPolygon = traffic[i].polygon;

    for (let j = 0; j < carPolygon.length; j++) {
      const touch = getIntersection(
        ray[0],
        ray[1],
        carPolygon[j],
        carPolygon[(j + 1) % carPolygon.length],
      );
      if (touch) {
        intersections.push(touch);
      }
    }
  }

  if (!intersections.length) {
    return null;
  }
  const touchOffsets = intersections.map((e) => e.offset);
  const minOffset = Math.min(...touchOffsets);

  return intersections.find((e) => e.offset === minOffset);
}
```

Код 3.11 – оновлення зчитувальних сенсорів

Нейронна мережа на кожній ітерації додатково вдосконалюється на якийсь відсоток (значення t). Відбувається це в коді за допомогою декількох циклів, що проходяться по кожному нейрону і лінійно інтерполюють їх порогові значення та вагу, поліпшуючи результат на деякий відсоток. В даному випадку останнє значення дорівнює 0.1, тобто 10%:

```
trainCars(cars);

/**
 * Get the stored best brain from the local storage, and train other cars.
 * @param {Car[]} cars
 * @param {number} trainValue
 */
export function trainCars(cars, trainValue = 0.1) {
  if (getBrainFromStorage()) {
    for (let i = 0; i < cars.length; i++) {
      cars[i].brain = getBrainFromStorage();

      if (i !== 0) {
        NeuralNetwork.train(cars[i].brain, trainValue);
      }
    }
  }
}
```

Код 3.12 – навчання нейронної мережі

Завдяки вбудованому в мову програмування JavaScript можливо створити «живу взаємодію» з використаними об’єктами – машинами. Дана функція має назву requestAnimationFrame та слугує для усіх видів робіт з анімаціями. Вона приймає, як параметр, деякий функціонал та виконує його тоді, коли браузер вважатиме за можливе здійснити поточну дію. Всі припустимі операції по застосуванню будь-якої оптимізації бере на себе браузер.

```
animate();

function animate(time) {
  update();

  bestCar = findBestCar(cars);

  display();

  networkCtx.lineDashOffset = -time / 50;
  NetworkVisualizer.visualize(networkCtx, bestCar.brain);

  requestAnimationFrame(animate);
}

function update() {
  for (let i = 0; i < traffic.length; i++) {
    traffic[i].update(road.borders, []);
  }
  for (let i = 0; i < cars.length; i++) {
    cars[i].update(road.borders, traffic);
  }
}

function display() {
  carCanvas.height = window.innerHeight;
```

```
  networkCanvas.height = window.innerHeight;

  carCtx.save();
  carCtx.translate(0, -bestCar.y + carCanvas.height * 0.7);

  road.display(carCtx);

  for (let i = 0; i < traffic.length; i++) {
    traffic[i].display(carCtx);
  }
  carCtx.globalAlpha = 0.2;

  for (let i = 0; i < cars.length; i++) {
    cars[i].display(carCtx);
  }
  carCtx.globalAlpha = 1;

  bestCar.display(carCtx, true);

  carCtx.restore();
}
```

Код 3.13 – анімація всіх об’єктів

Функціонал, який виконує браузер при кожній ітерації анімації, включає в себе оновлення координат трафіку і автівок, а також ширини та висоти простору навколо, знаходження найкращої машини для нейронної мережі, а також актуалізація та візуалізація поточних дій штучної мережі.

Підсумовуючи, роботу над додатком можливо розподілити на декілька етапів, що поступово доповнюють один одного та додають більше і більше функціоналу до проекту. Таблиця, що наведена нижче, більш детально описує усі необхідні стадії, та їх відповідні рішення засобами мови програмування JavaScript, для виконання даної дослідницької роботи. Вона також підсумовує всю наведену інформацію щодо впровадження функцій, які були описані в цьому розділі.

Таблиця 3.1 – етапи розробки системи

Етап	Рішення
Впровадження механіки управління автомобілем	Клас «Controls» з відповідними полями forward, left, right, reverse.

Визначення дороги	Клас «Road», що має поля, які відповідають за ширину, кількість дорожніх ліній, ліва та права границі, їх бордюрів, а також математичні кутки з відповідними координатами.
Створення зчитувальних сенсорів	Декілька класів «Sensor», які мають визначене авто, до якого приєднані, а також поля, що представляють кількість сенсорів, їх довжину та відстань один від одного, координати, а також результати зчитувань.
Реалізація виявлення зіткнень	Функція для створення чотирьох точок полігону. При кожному кадровому оновлені, відбувається перевірка циклом на наявність колізії між усіма точками бордюр та авто. Це можливо завдяки функції виявлення перетину двох об’єктів. В такому випадку контроль нейронної мережі переходить до іншої машини.
Симуляція трафіку	Генерація на початку роботи додатку додаткових авто, що не керуються нейронною мережею, а «просто є» та рухаються по дорозі вперед. Вони враховуються зчитувальними сенсорами та при колізії.
Розробка нейронної мережі	Клас «Neural Network», що має масив нейронів як вхідний параметр, а також містить поле у вигляді масиву рівнів. Останні мають свій власний клас «Level», і приймають кількість нейронів для вхідного та вихідного шарів. Мають функції алгоритму прямого поширення, вдосконалення мережі, і випадкова генерація значень нейронів.
Паралельність	Вдосконалення головної функції кадрового оновлення системи, в якій циклом будуть «освіжатись» дані про усі згенеровані авто. Останні

	включають в себе як ті, що не керуються нейронною мережею, так і ті, що намагаються доїхати до кінця.
Генетичний алгоритм	Функція, яка дозволяє «мутацію» нейронної мережі на якийсь відсоток, в кращу сторону. Відбувається проходження циклом по кожному рівню та удосконалення порогового значення та ваги всякого нейрону, завдяки формулі лінійної інтерполяції.

3.3 Розроблені алгоритми

Для реалізації машинного навчання необхідно створити одну із вищезгаданих типів нейронних мереж та обрати алгоритм, за допомогою якого буде відбуватись її тренування.

Як було вже згадано раніше, в даній дослідницькій роботі було вирішено обрати нейронну мережу типу прямого поширення. Це обумовлено тим, що перед нами стоїть задача навчити штучну мережу розпізнавати об’єкти та старатись реагувати відповідно. Таким чином, застосовуючи даний тип мережі, буде втілюватись її основний алгоритм – подання інформації «вперед».

При початковому запуску програми, у кожної із згенерованих авто, але не тих, що створюють імітацію трафіку, створюється так званий «мозок», що і відповідає за нейронну мережу та її навчання. Вона формується із масиву, що означає кількість нейронів на кожному із рівнів мережі, включаючи початковий та кінцевий.

Як тільки перший етап формування мережі та її рівнів завершений, відбувається випадкова генерація ваг деякого шару, що є поточним на ітераційному циклі. Для кожного нейрону на поточному рівні, присвоюється створена вага під всякий нейрон наступного шару, в таких межах: $w \in (-1; 1)$. Тобто відбудеться подвійний ітераційний цикл, який «пройдеться» по всім

вхідним та вихідним нейронам і присвоїть їм деяку вагу для подальших обчислень. Чому саме від (-1; 1)? Від’ємне значення буде означати, що саме в ту сторону, де є така цифра, нейронна мережа не повинна повертати, адже це буде або зіткнення, або з’їзд з дороги. Позитивні ж моменти навпаки свідчать, що необхідно зробити крок саме в даний бік, і ніяк інакше. На рисунку нижче негативні цифра позначені синім кольором, а позитивні – жовтим, і тому нейронна мережа «натискає» саме ці клавіші.

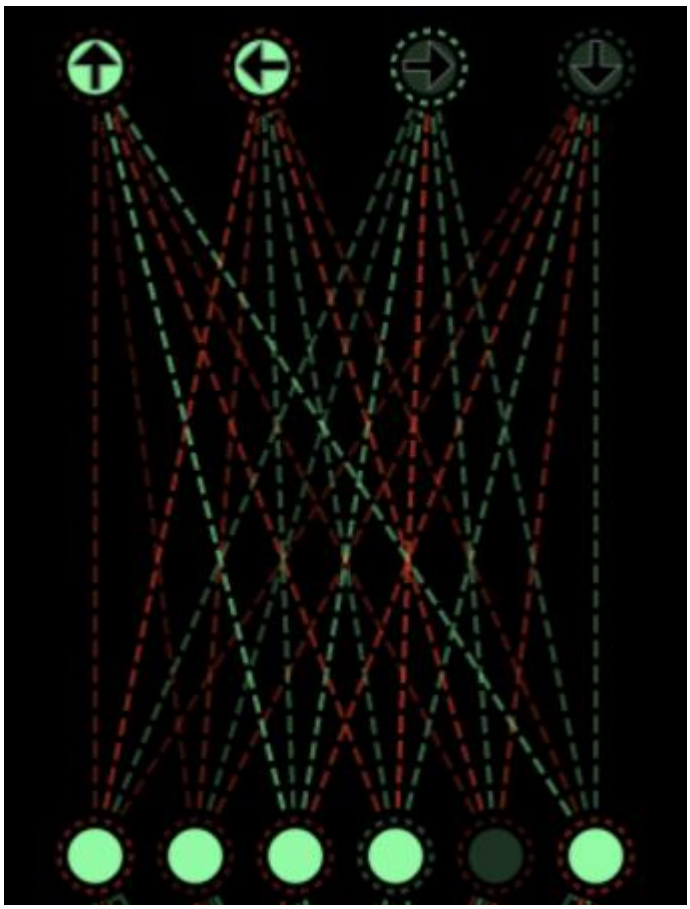

Рисунок 3.7 – відображення ваги нейронів

Основний алгоритм нейронної мережі прямого поширення полягає в тому, що вона не утворює циклів між її рівнями, але передає всю інформацію на наступний рівень, від кожного нейрону поточного шару до всякого нейрону наступного. Це також було показано на рисунку вище. За допомогою циклу відбувається підсумовування множення поточного вхідного нейрону та ваги його вихідного нейрону.

$$\text{Сума вихідного нейрону} = \sum_{i=0}^{n} x_i w_{ji},$$

де i – кількість під’єднаних вихідних нейронів, j – кількість вхідних нейронів, x – значення поточного вхідного нейрону, а w – вага взятого вихідного нейрону.

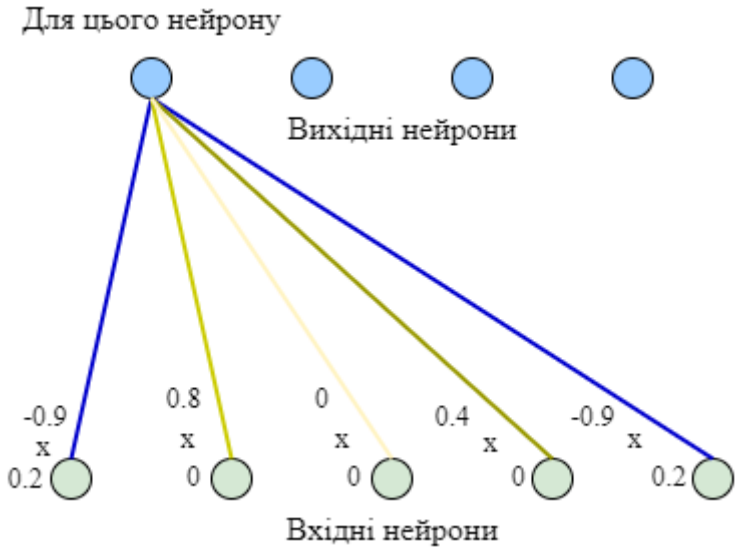

Рисунок 3.8 – розрахунок суми вихідного нейрону

Після того, як для кожного вихідного нейрону була розрахована сума, вона, як було згадано раніше, по формулі повинна зрівнятися з пороговим значенням (або зсувом) об’єкта, після чого визначається що більше. Якщо підсумок більше, ніж зсув нейрону, тоді присвоюється 1 як значення обраного вихідного нейрону, інакше – виставляється 0. Це важливі цифри, адже від них залежать подальші дії нейронної мережі. В залежності від того, яке значення було присвоєно вихідному нейрону, штучна мережа через пряме поширення «передає» наперед отриману інформацію та натискає віртуальні клавіші в залежності від того, який нейрон має одиницю. Це дозволяє керувати авто та в цілому виконувати саме ті функції, які необхідні для здійснення даної дослідницької роботи.

Кожного разу, коли відбувається покадрове оновлення всіх авто, включаючи їх координати на полотні, відбувається перерахунок вихідних нейронів штучної мережі, від яких динамічно змінюється рух «головної»

машини. Це також корегується таким параметром, як відстань об’єкту від зчитувальних сенсорів, або центру, що присутні на провідному авто. Вони виступають провідником інформації щодо того, що чекає на об’єкт попереду: чим ближче до нього інша машина, тим сильніше і більш зрозуміло які дії необхідно зробити. Аналогією до цього методу буде світіння ліхтарика на стелю – при меншому ступені близькості від буде яскравіше, і навпаки.

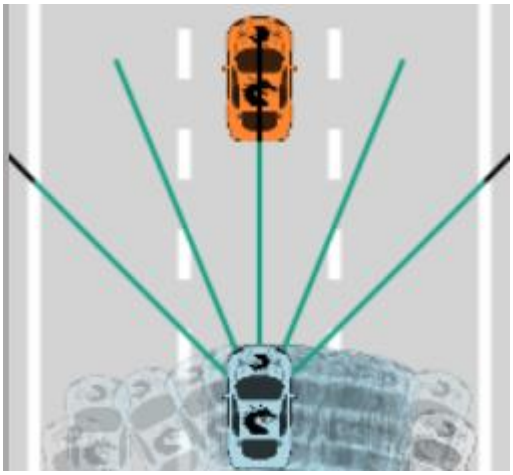

Рисунок 3.9 – зчитувальні сенсори авто

Зчитувальні сенсори відтворюються кожного разу, як нейронна мережа отримує під контроль будь-яке із автівок. Елементи інформаційного характеру отримують дані через відстань об’єкта від дослідницької машини. Це реалізовано завдяки функції перетину з двома відрізками, що мають свої координати початку та кінця.

Перетин двох відрізків, зокрема, визначається за допомогою формули лінійної інтерполяції, яка є дуже зручним визначенням та часто використовується при розробці ігор. Вона інтерполює значення в діапазоні [від...до] на основі деякого третього параметру, що зазвичай називається t. Остання цифра означає наскільки об’єкт далеко від першого параметру як відсоткове значення, який є одним із двох цифр інтерполяції, що репрезентують місце знаходження предмета по осі координат x.

$$y = a + (b - a) * t$$

Тобто при $t = 0.5$, результуюче значення по осі координат x буде посередині між точкою A та точкою B. Саме ця формула лежить у визначенні відстані від початку авто до перетину з будь-яким об’єктом попереду.

Для знаходження перетину двох об’єктів (наприклад відрізків), необхідно скористатися деякою формулу, яка включає в себе застосування лінійної інтерполяції. Нехай A та B – дві точки деякого відрізку, C та D – дві точки другого відрізку, що перетинається з першим. Тоді I – це точка перетину цих двох об’єктів, а його координати вираховуються за такими формулами:

$$\begin{cases} I_x = A_x + (B_x - A_x)t = C_x + (D_x - C_x)u \\ I_y = A_y + (B_y - A_y)t = C_y + (D_y - C_y)u \end{cases}$$

Після скорочень, рівняння приводиться до такого вигляду:

$$t = \frac{(D_x - C_x)(A_y - C_y) - (D_y - C_y)(A_x - C_x)}{(D_y - C_y)(B_x - A_x) - (D_x - C_x)(B_y - A_y)}$$

Потім перетин двох об’єктів (перетинів) розраховується за допомогою лінійної інтерполяції, що повертає координати деякої точки посередині (x та y):

$$x = A_x + (B_x - A_x) * t$$
$$y = A_y + (B_y - A_y) * t$$

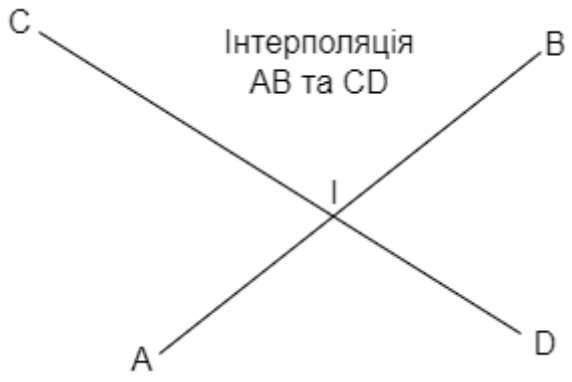

Рисунок 3.10 – перетин двох відрізків

Завдяки отриманню нових значень, що поступають від зчитувальних сенсорів, нейронна мережа з’ясовує, що відбувається попереду та від цього

динамічно робить свій рух у вигляді тренування через алгоритм прямого поширення. Зрештою все прийде до того, що на виході вийде чотири значення, які відображають інформацію щодо того, чи потрібно натиснути нейронній мережі ту чи іншу клавішу руху авто.

Виявлення колізії з другими об’єктами виконується за допомогою створення та отримання математичним шляхом чотирьох точок машини, які потім застосовуються для прояву факту зіткнення із бордюрами дороги або іншими авто. Тут також залучений вищезгаданий функціонал перетину двох об’єктів, який виконує функцію знаходження свідчення про те, чи перетнула головна машина бордюри шосе або інші автомобілі. Таким чином, при відбуванні будь-якої колізії, авто вважається «пошкодженим» та перестає існувати в межах програми, а управління над ним переходить до іншої машини.

Алгоритм лінійної інтерполяції та перетину добре працюють, але є ще одна деталь, яка відсутня та потребує втручання. Це механізм самого навчання та покращення результатів попередньої ітерації нейронної мережі. Тому для цього була створена ще одна дія, функція якої – взяти усі порогові значення всякого нейрону із кожного рівня штучної мережі та вдосконалити (добавити) до них зверху ще якусь цифру. Яку саме – вирішить, знову ж таки, лінійна інтерполяція, що приймає як початковий параметр поточний зсув нейрону, або його вагу (виконується дві ітерації), а як другий – випадкове значення від -1 до 1, та t, яке буде приймати відсоткове значення. В залежності від останнього, дослідницька нейронна мережа буде удосконалюватись, відштовхуючись від початкового значення поточного нейрону кожного рівня, тим самим відрізняючись від мережі, що була на попередньому ітераційному циклі навчання.

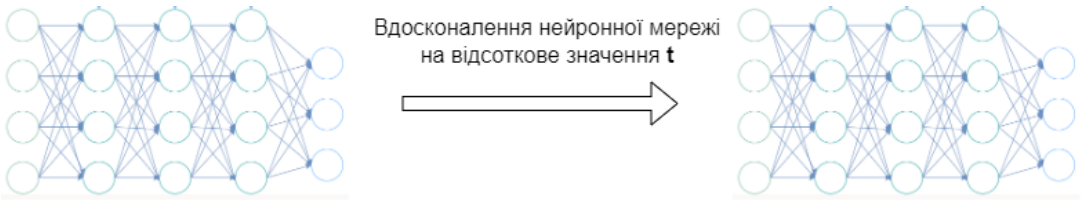

Рисунок 3.11 – мутація нейронної мережі на t

3.4 Результати роботи програми

Результатом виконання розробки програмного забезпечення під потреби даної дослідницької роботи став повноцінний веб-сайт, який створений та візуалізований на мові програмування JavaScript. Це дає змогу легко інтегрувати додаток як веб-сервіс, продемонструвати його можливості, а також проводити дії, пов'язані зі спостереженням та дослідженням поведінки нейронної мережі під потреби авторів. [28]

Програма перед початком та відтворенням усіх основних елементів, «підготовлює» полотно для дороги, на якій будуть відбуватись вже знайомі наступні події. Ще один такий простір створюється для відображення поведінки нейронної мережі, мета якої – тренування та навчання.

Остання має декілька рівнів, в які входять вхідний та вихідний, що характеризують та описують її практику, а також нейрони на кожному із шарів. Вони мають свою власну вагу та порогове значення, завдяки яким мають змогу управляти однією із створених на початку програми автівок.

Розмовляючи про останні, їх генерація відбувається щойно, як завершаються попередні два етапи, і розпочинається наступна. На вхід до функції подається певний параметр – деяке число – а після відбувається створення саме такої кількості машин.

Базуючись на розглянутих математичних формулах та алгоритмах, нейронна мережа запрограмована таким чином, щоб мати вихідний рівень із чотирма нейронами, які відповідають за рух авто в одну із чотирьох сторін. Допомога з вибором також надходить від зчитувальних сенсорів – спеціальних

променів, що через лінійну інтерполяцію вираховують відстань до об’єкту попереду та надають сигнал, стимулюючий до певних дій.

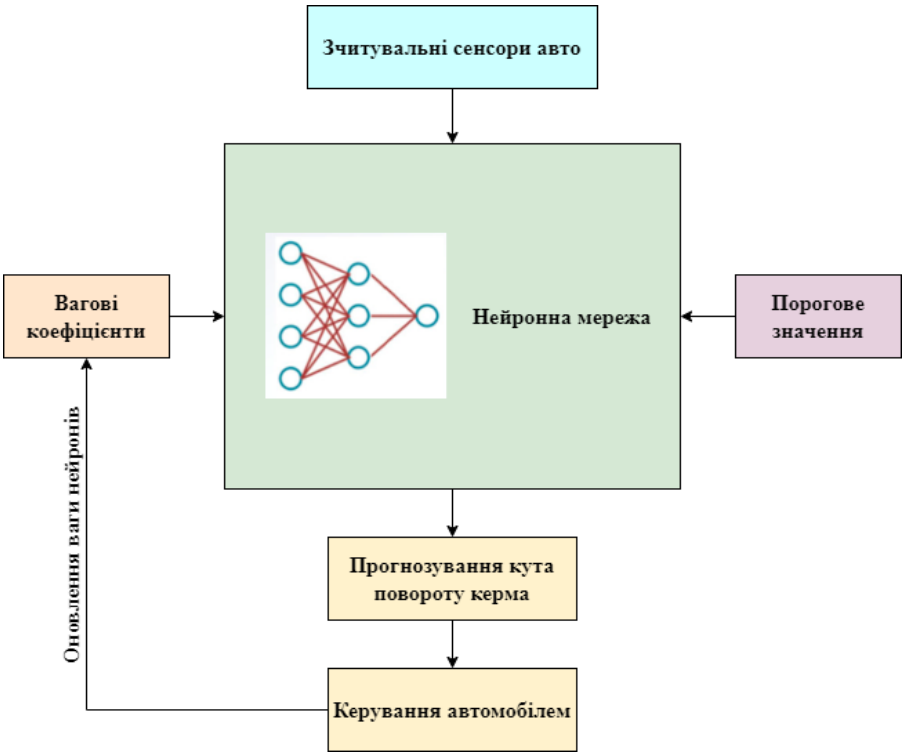

Рисунок 3.12 – життєвий цикл нейронної мережі

Операції та сигнали на їх виконання від нейронної мережі обов’язково потрібні, адже попереду також їдуть додаткові машини, які теж генеруються на початку роботи програми, але нічим не контролюються. Їх створення є умисним, з метою тренування дослідницької мережі, та відбувається на кожній із можливих дорожніх полос.

Бувають і невдачі – нейронна мережа не може завжди уникати пошкоджень, а тому має властивість інколи стикатися з іншими об’єктами. Коли це відбувається, таке авто вважається дефектним, а тому вибуває з роботи програми. Під час цього процесу, якщо нейронна мережа управляла машиною, то програма запрограмована передати керування до іншої. Остання програмно обирається по параметру найбільшого просування по дорозі вперед, а якщо бути точнішим – по її координатам, зокрема у.

В кінці буває лише два результати. Перший – це коли усі авто не змогли «впоратись з управлінням» та зіткнулися з іншими об’єктами – програш.

Інший же являє собою перемогу і наслідок усіх зусиль нейронної мережі, та її тренувань і навчань. Вона змогла достатньо удосконалитись, щоб доїхати до кінця без перешкод.

ВИСНОВКИ

Результатом виконання даної дослідницької роботи стало створення інформаційної технології – аналітичного веб-додатку – який дозволяє робити спостереження над навчанням нейронної мережі типу прямого поширення. Це призводить до того, що у користувача системи є можливість досліджувати всі дії штучної мережі та те, як вона працює та тренується, у вигляді веб-сайту, написаному завдяки гіпертекстової розмітки HTML, каскадних стилів CSS, та мові програмування JavaScript.

Під час роботи над даним дипломним проектом було проведено огляд існуючих рішень, який складався з вибору типу необхідної саме для цієї проблеми нейронної мережі та застосовуваних до неї алгоритмів. Також було досліджено декілька мов програмування, що найбільше підходять для завдань, де залучене машинне навчання, та обрано одну із них, що є найбільш оптимальною в рамках цього дослідження. Окрім цього було оглянуто як саме працюють нейронні мережі та глибоке навчання, які алгоритми та формули застосовують, та як відбувається взаємодія. Після огляду існуючих рішень та методів реалізації було проведено проектування та формування вхідних даних, разом із розробкою основних застосовуваних алгоритмів у роботі. На завершальному етапі було проведено програмну реалізацію.

Продуктом виконання даного дослідження став програмний застосунок, який має такі результуючі показники: при N = 100 (кількість згенерованих авто, що управляються нейронною мережею) та 5 зчитувальних променів сенсорів авто, кожний з яких має дальність у 150 пікселів, нейронна мережа змогла навчитись та виконати усі поставлені завдання за 17 генерацій. Кожна ітерація навчала та покращувала досвід мережі на t = 0.1, тобто 10%. Штучна мережа мала 3 рівні: вхідний складався із 5 нейронів, внутрішній мав 6, а вихідний – 4. Із цього випливає, що подальші тренування, починаючи з даного покоління, не потрібні, а завдання – виконане. Перший випробувальний етап був в середньому пройдений за 14 секунд, а фінальний – за 18. Це означає, що

на тренування штучної мережі, при таких вхідних даних та програмних показниках, в цілому витратилось 244 секунди, або 4 хвилини і 4 секунди.

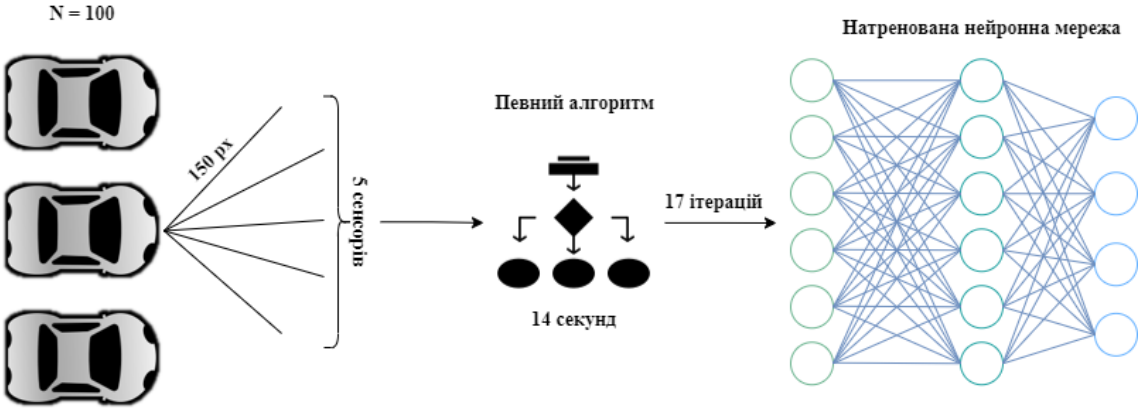

Рисунок – результуючі показники

Випливаючи із результатів виконання роботи, було створено програмне середовище, що має можливість:

- відтворювати каркаси для відображення шосе та дій самої штучної мережі;
- генерувати деяку кількість вхідних авто для їх подальшого навчання;
- створювати імітацію трафіку на дорозі;
- реалізовувати алгоритм нейронної мережі, який буде управляти машинами;
- програмно обирати «головну» машину та віддавати контроль над нею нейронній мережі;
- втілювати недопустимість колізій між об’єктами та видалити пошкоджене авто в такому разі;
- запрограмувати передачу нейронній мережі контролю над авто до другої при зіткненні;
- реалізовувати навчання даних нейронної мережі з кожною новою ітерацією;
- створювати збереження та видалення інформації про штучну мережу на поточній послідовності.

СПИСОК ЛІТЕРАТУРИ

1. Trask A. Grokking Deep Learning / A. Trask. – Manning Publications, 2019. – P. 4 – 175.
2. Buduma N. Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms / N. Buduma. – O'Reilly Media, 2017. – P. 1 – 37, 63 – 115.
3. Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain / F. Rosenblatt. – 1958.
4. Balas E. V. Multilayer Perceptron and Neural Networks / E. V. Balas, Mastorakis E. N. – 2009. – 11 p.
5. Sazli M. A brief review of feed-forward neural networks / M. H. Sazli. – 2006. – 8 p.
6. Venkatesan R. Convolutional Neural Networks in Visual Computing: A Concise Guide / R. Venkatesan, Li B. – Phoenix: CRC Press, 2018. – P. 89 – 117.
7. Du K. Neural Networks and Statistical Learning / K. Du, M.N.s Swamy. – Montreal, 2014. – P. 312 – 319, 351 – 363.
8. Kamel M. S. Modular neural network architectures for classification / M. S. Kamel, H. Rafaat. – 1996. – 7 p.
9. Stewart A. Python Programming for Beginners: A Comprehensive Guide to Learning the Basics of Python Programming / A. Stewart. – 2016. – P. 1 – 15.
10. Zocca V. Python Deep Learning: Next generation techniques to revolutionize computer vision, AI, speech and data analysis / V. Zocca, Spacagna G., Slater D., Roelants P. – Packt Publishing, 2017. – P. 37 – 98.
11. Rauschmayer A. JavaScript for impatient programmers / A. Rauschmayer. – 2019. – P. 19 – 30, 37 – 364.
12. Kanber B. Hands-on Machine Learning with JavaScript: Solve complex computational web problems using machine learning / B. Kanber – P. 29 – 115.

13. Mailund T. Advanced Object-Oriented Programming in R: Statistical Programming for Data Science, Analysis and Finance / T. Mailund. – Apress, 2017. – P. 1 – 35.
14. Wiley J. R Deep Learning Essentials: Build Automatic Classification and Prediction Models Using Unsupervised Learning / J. Wiley. – Birmingham: Packt Publishing, 2016. – P. 8 – 37.
15. Sanderson S. Java for Beginners: a simple start to Java programming / S. Sanderson. – 2019. – P. 4 – 114.
16. Sugomori Y. Java Deep Learning Essentials / Y. Sugomori. – Packt Publishing, 2016. – P. 143 – 177.
17. Richard G. Beginning C++ Programming / G. Richard. – Birmingham: Packt Publishing, 2017. – P. 1 – 9.
18. Kolodiazhnyi K. Hands-On Machine Learning with C++: Build, train, and deploy end-to-end marchine learning and deep learning pipelines / K. Kolodiazhnyi. – Birmingham: Packt Publishing, 2020. – P. 12 – 164, 313 – 376.
19. McCulloh W. A logical calculus of the ideas immanent in nervous activity / W. McCulloh, W. Pitts. – 1943. – 17 p.
20. Werbos P. Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences / P. Werbos. – Cambridge. – 1974.
21. LeCun Y. Backpropagation Applied to Handwritten Zip Code Recognition / Y. Lecun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackeld. – 1989. – 11 p.
22. Ayeni A. W. Empirics of Standard Deviation / A. W. Ayeni. – 2014. – 8 p.
23. Goodfellow I. Gradient Descent and the Structure of Neural Network Cost Functions / I. Goodfellow. – 2015. – P. 4 – 19.
24. Rosenthal S. Linear Regression Analysis / S. Rosenthal. – P. 1 – 9.
25. Frain B. Responsive Web Design with HTML5 and CSS3: Build responsive and future-proof websites to meet the demands of modern web users / B. Frain. – Birmingham: Packt Publishing, 2015. – P. 77 – 92.

26. Meyer E. CSS: The Definite Guide: Visual Presentation for the Web / E. Meyer, Weyl E. – Sebastopol: O’Reilly, 2018. – P. 38 – 45, 315 – 385 387 – 396, 530 – 534, 569 – 609.
27. Marrs T. JSON at Work: Practical Data Integration for the Web / T. Marrs. – Sebastopol: O’Reilly, 2017. – P. 25 – 35, 113 – 146.
28. Dale K. Data Visualization with Python and JavaScript: Crafting a Data-visualisation Toolchain for the Web / K. Dale. – Sebastopol: O’Reilly, 2016. – P. 105 – 142.
29. Camden R. Client-Side Data Storage: Keeping It Local / R. Camden. – Sebastopol: O’Reilly Media, 2016. – P. 1 – 27.

ДОДАТКИ

Додаток А. Початковий скрипт

```
import { Road } from './road';
import { Car, findBestCar, generateCars } from './car';
import { getRandomColor, removeBrainFromStorage, saveBrainToStorage } from
'./utils';
import { trainCars } from './network';
import { NetworkVisualizer } from './network-visualizer';

window.save = saveBrainToStorage;
window.discard = removeBrainFromStorage;

/**
 * @type {HTMLCanvasElement}
 */
const carCanvas = document.getElementById('car-canvas');
carCanvas.width = 200;

/**
 * @type {HTMLCanvasElement}
 */
const networkCanvas = document.getElementById('network-canvas');
networkCanvas.width = 300;

const carCtx = carCanvas.getContext('2d');
const networkCtx = networkCanvas.getContext('2d');

const road = new Road(carCanvas.width / 2, carCanvas.width * 0.9);

const cars = generateCars(road, 'AI', 100);
let bestCar = cars[0];

/**
 * @type {Car[]}
 */
const traffic = [
  new Car(road.getLineCenter(1), -100, 30, 50, 'DUMMY', 2, getRandomColor()),
  new Car(road.getLineCenter(0), -300, 30, 50, 'DUMMY', 2, getRandomColor()),
  new Car(road.getLineCenter(2), -300, 30, 50, 'DUMMY', 2, getRandomColor()),
  new Car(road.getLineCenter(0), -500, 30, 50, 'DUMMY', 2, getRandomColor()),
  new Car(road.getLineCenter(1), -500, 30, 50, 'DUMMY', 2, getRandomColor()),
  new Car(road.getLineCenter(1), -700, 30, 50, 'DUMMY', 2, getRandomColor()),
  new Car(road.getLineCenter(2), -700, 30, 50, 'DUMMY', 2, getRandomColor()),
];

trainCars(cars);

animate();

function animate(time) {
  update();

  bestCar = findBestCar(cars);

  display();

  networkCtx.lineDashOffset = -time / 50;
  NetworkVisualizer.visualize(networkCtx, bestCar.brain);

  requestAnimationFrame(animate);
```

```
}

function update() {
  for (let i = 0; i < traffic.length; i++) {
    traffic[i].update(road.borders, []);
  }
  for (let i = 0; i < cars.length; i++) {
    cars[i].update(road.borders, traffic);
  }
}

function display() {
  carCanvas.height = window.innerHeight;
  networkCanvas.height = window.innerHeight;

  carCtx.save();
  carCtx.translate(0, -bestCar.y + carCanvas.height * 0.7);

  road.display(carCtx);

  for (let i = 0; i < traffic.length; i++) {
    traffic[i].display(carCtx);
  }
  carCtx.globalAlpha = 0.2;

  for (let i = 0; i < cars.length; i++) {
    cars[i].display(carCtx);
  }
  carCtx.globalAlpha = 1;

  bestCar.display(carCtx, true);

  carCtx.restore();
}
```

Додаток Б. Скрипт класу авто

```
import { Road } from './road';
import { Sensor } from './sensor';
import { NeuralNetwork } from './network';
import { Controls } from './controls';
import { isPolygonIntersected } from './utils';

const imgSrc = './assets/car.png';
const ctxDestination = 'destination-atop';

export class Car {
  /**
   * @param {number} x x
   * @param {number} y y
   * @param {number} width The width of a car.
   * @param {number} height The height of a car.
   * @param {'DUMMY' | 'AI'} type The type of car: for making traffic or for
training.
   * @param {number} maxSpeed The maximum speed of a car. The default value
is 3.
   * @param {string} color The color of a car. The default is blue.
   */
  constructor(x, y, width, height, type, maxSpeed = 3, color = 'lightblue') {
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.color = color;

    this.speed = 0;
    this.maxSpeed = maxSpeed;
    this.acceleration = 0.2;
    this.friction = 0.05;
    this.angle = 0;

    this.isDestroyed = false;

    if (type === 'AI') {
      this.sensor = new Sensor(this);
      this.brain = new NeuralNetwork(
        [this.sensor.raysCount, 6, 4],
      );
    }
    this.controls = new Controls(type);

    this.#loadCarImage();
  }

  #loadCarImage() {
    this.image = new Image();
    this.image.src = imgSrc;

    this.carCanvas = document.createElement('canvas');
    this.carCanvas.width = this.width;
    this.carCanvas.height = this.height;

    const canvasCtx = this.carCanvas.getContext('2d');

    this.image.onload = () => {
      canvasCtx.fillStyle = this.color;
      canvasCtx.rect(0, 0, this.width, this.height);
      canvasCtx.fill();
```

```
    canvasCtx.globalCompositeOperation = ctxDestination;
    canvasCtx.drawImage(this.image, 0, 0, this.width, this.height);
  };
}

#move() {
  if (this.controls.forward) {
    this.speed += this.acceleration;
  }

  if (this.controls.backward) {
    this.speed -= this.acceleration;
  }

  if (this.speed > this.maxSpeed) {
    this.speed = this.maxSpeed;
  }

  if (this.speed < -this.maxSpeed / 2) {
    this.speed = -this.maxSpeed / 2;
  }

  if (this.speed > 0) {
    this.speed -= this.friction;
  }

  if (this.speed < 0) {
    this.speed += this.friction;
  }

  if (Math.abs(this.speed) < this.friction) {
    this.speed = 0;
  }

  if (this.speed !== 0) {
    const flip = this.speed > 0 ? 1 : -1;

    if (this.controls.left) {
      this.angle += 0.03 * flip;
    }

    if (this.controls.right) {
      this.angle -= 0.03 * flip;
    }
  }

  this.x -= Math.sin(this.angle) * this.speed;
  this.y -= Math.cos(this.angle) * this.speed;
}

#createPolygon() {
  /**
   * @type {{x: number, y: number}[]}
   */
  const points = [];
  const radius = Math.hypot(this.width, this.height) / 2;
  const alphaAngle = Math.atan2(this.width, this.height);

  points.push({
    x: this.x - Math.sin(this.angle - alphaAngle) * radius,
    y: this.y - Math.cos(this.angle - alphaAngle) * radius,
  });
```

```
    points.push({
      x: this.x - Math.sin(this.angle + alphaAngle) * radius,
      y: this.y - Math.cos(this.angle + alphaAngle) * radius,
    });
    points.push({
      x: this.x - Math.sin(Math.PI + this.angle - alphaAngle) * radius,
      y: this.y - Math.cos(Math.PI + this.angle - alphaAngle) * radius,
    });
    points.push({
      x: this.x - Math.sin(Math.PI + this.angle + alphaAngle) * radius,
      y: this.y - Math.cos(Math.PI + this.angle + alphaAngle) * radius,
    });

    return points;
  }

  /**
   * @param {{ x: number, y: number }[][]} roadBorders
   * @param {Car[]} traffic
   */
  #isIntersected(roadBorders, traffic) {
    for (let i = 0; i < roadBorders.length; i++) {
      if (isPolygonIntersected(this.polygon, roadBorders[i])) {
        return true;
      }
    }
    for (let i = 0; i < traffic.length; i++) {
      if (isPolygonIntersected(this.polygon, traffic[i].polygon)) {
        return true;
      }
    }
    return false;
  }

  /**
   * Correctly display the car on the road on the canvas context.
   * @param {CanvasRenderingContext2D} ctx
   * @param {boolean} isSensor If the car have its sensor to display.
   */
  display(ctx, isSensor = false) {
    if (this.sensor && isSensor) {
      this.sensor.display(ctx);
    }

    ctx.save();
    ctx.translate(this.x, this.y);
    ctx.rotate(-this.angle);

    if (!this.isDestroyed) {
      ctx.drawImage(
        this.carCanvas,
        -this.width / 2,
        -this.height / 2,
        this.width,
        this.height,
      );
      ctx.globalCompositeOperation = 'multiply';
    }

    ctx.drawImage(
      this.image,
      -this.width / 2,
      -this.height / 2,
```

```
      this.width,
      this.height,
    );

    ctx.restore();
  }

  /**
   * A function to update the car according to the neural network.
   * @param {{ x: number, y: number }[][]} roadBordersCoords
   * @param {Car[]} traffic
   */
  update(roadBordersCoords, traffic) {
    if (!this.isDestroyed) {
      this.#move();

      this.polygon = this.#createPolygon();
      this.isDestroyed = this.#isIntersected(roadBordersCoords, traffic);
    }
    if (this.sensor) {
      this.sensor.update(roadBordersCoords, traffic);

      const offsets = this.sensor.readings.map(
        (reading) => (reading ? 1 - reading.offset : 0),
      );
      const [forward, left, right, backward] =
NeuralNetwork.feedForward(offsets, this.brain);

      if (this.brain) {
        this.controls.forward = forward;
        this.controls.left = left;
        this.controls.right = right;
        this.controls.backward = backward;
      }
    }
  }
}

/**
 * Generates and returns a number of cars.
 * @param {Road} road A road to build cars on.
 * @param {'DUMMY' | 'AI'} type The type of cars.
 * @param {number} n A number to create.
 * @return {Car[]} The created cars.
 */
export function generateCars(road, type, n = 1) {
  const cars = [];

  for (let i = 0; i < n; i++) {
    cars.push(new Car(road.getLineCenter(1), 100, 30, 50, type));
  }

  return cars;
}

/**
 * Find the best car from all the generated.
 * @param {Car[]} cars
 */
export function findBestCar(cars) {
  return cars.find((car) => car.y === Math.min(...cars.map((c) => c.y)));
}
```

Додаток В. Скрипт класу дороги

```
import { lerp } from './utils';

const infiniteValue = 1000000;

export class Road {
  /**
   * @param {number} x A value to center the road around
   * @param {number} width A width value of the road
   * @param {number} linesCount The road lines count (default is 3)
   */
  constructor(x, width, linesCount = 3) {
    this.x = x;
    this.width = width;
    this.linesCount = linesCount;

    this.left = x - width / 2;
    this.right = x + width / 2;
    this.top = -infiniteValue;
    this.bottom = infiniteValue;

    const topLeft = {
      x: this.left,
      y: this.top,
    };
    const topRight = {
      x: this.right,
      y: this.top,
    };
    const bottomLeft = {
      x: this.left,
      y: this.bottom,
    };
    const bottomRight = {
      x: this.right,
      y: this.bottom,
    };
    this.borders = [
      [topLeft, bottomLeft],
      [topRight, bottomRight],
    ];
  }

  /**
   * Correctly display the road on a canvas context.
   * @param {CanvasRenderingContext2D} ctx The canvas context.
   */
  display(ctx) {
    ctx.lineWidth = 5;
    ctx.strokeStyle = 'white';

    for (let i = 1; i <= this.linesCount - 1; i++) {
      const x = lerp(
        this.left,
        this.right,
        i / this.linesCount,
      );

      ctx.setLineDash([20, 20]);
      ctx.beginPath();
      ctx.moveTo(x, this.top);
      ctx.lineTo(x, this.bottom);
```

```
      ctx.stroke();
    }
    ctx.setLineDash([]);

    this.borders.forEach((border) => {
      ctx.beginPath();

      ctx.moveTo(border[0].x, border[0].y);
      ctx.lineTo(border[1].x, border[1].y);

      ctx.stroke();
    });
  }

  /**
   * Get the center of a road line.
   * @param {number} lineNum
   * @return {number} x
   */
  getLineCenter(lineNum = 1) {
    const lineWidth = this.width / this.linesCount;
    return this.left + lineWidth / 2
      + Math.min(lineNum, this.linesCount - 1) * lineWidth;
  }
}
```

Додаток Г. Скрипт зчитувальних сенсорів

```
// eslint-disable-next-line import/no-cycle
import { Car } from './car';
import { lerp, getIntersection } from './utils';

export class Sensor {
  /**
   * A reading sensor of a car. It has 3 rays (`Sensor.rays`) to look for
objects.
   * @param {Car} car The car.
   */
  constructor(car) {
    this.car = car;

    this.raysCount = 5;
    this.raysLength = 150;
    this.raysSpreadWidth = Math.PI / 2;

    /**
     * @type {[{x: number, y: number}, {x: number, y: number}][]}
     */
    this.rays = [];
    /**
     * @type {{x: number, y: number, offset: number}[]}
     */
    this.readings = [];
  }

  /**
   * A function to update the sensors of a car.
   * @param {[{x: number, y: number}, {x: number, y: number}][]}
roadBordersCoords
   * @param {Car[]} traffic
   */
  update(roadBordersCoords, traffic) {
    this.#createRays();

    this.readings = [];

    for (let i = 0; i < this.rays.length; i++) {
      this.readings.push(
        Sensor.#getReading(
          this.rays[i],
          roadBordersCoords,
          traffic,
        ),
      );
    }
  }

  #createRays() {
    this.rays = [];

    for (let i = 0; i < this.raysCount; i++) {
      const rayAngle = lerp(
        this.raysSpreadWidth / 2,
        -this.raysSpreadWidth / 2,
        this.raysCount === 1 ? 0.5 : i / (this.raysCount - 1),
      ) + this.car.angle;

      const start = {
        x: this.car.x,
```

```
      y: this.car.y,
    };
    const end = {
      x: this.car.x - Math.sin(rayAngle) * this.raysLength,
      y: this.car.y - Math.cos(rayAngle) * this.raysLength,
    };

    this.rays.push([start, end]);
  }
}

/**
 *
 * @param {[{x: number, y: number}, {x: number, y: number}]} ray
 * @param {{ x: number, y: number }[][]} roadBorders
 * @param {Car[]} traffic
 */
static #getReading(ray, roadBorders, traffic) {
  const intersections = [];

  for (let i = 0; i < roadBorders.length; i++) {
    const touch = getIntersection(
      ray[0],
      ray[1],
      roadBorders[i][0],
      roadBorders[i][1],
    );
    if (touch) {
      intersections.push(touch);
    }
  }

  for (let i = 0; i < traffic.length; i++) {
    const carPolygon = traffic[i].polygon;

    for (let j = 0; j < carPolygon.length; j++) {
      const touch = getIntersection(
        ray[0],
        ray[1],
        carPolygon[j],
        carPolygon[(j + 1) % carPolygon.length],
      );
      if (touch) {
        intersections.push(touch);
      }
    }
  }

  if (!intersections.length) {
    return null;
  }
  const touchOffsets = intersections.map((e) => e.offset);
  const minOffset = Math.min(...touchOffsets);

  return intersections.find((e) => e.offset === minOffset);
}

/**
 * Display the sensor rays on a canvas context.
 * @param {CanvasRenderingContext2D} ctx
 */
display(ctx) {
  for (let i = 0; i < this.raysCount; i++) {
```

```
      let endPoint = this.rays[i][1];

      if (this.readings[i]) {
        endPoint = this.readings[i];
      }

      ctx.beginPath();
      ctx.lineWidth = 2;
      ctx.strokeStyle = 'rgb(22, 160, 133)';
      ctx.moveTo(
        this.rays[i][0].x,
        this.rays[i][0].y,
      );
      ctx.lineTo(
        endPoint.x,
        endPoint.y,
      );
      ctx.stroke();

      ctx.beginPath();
      ctx.lineWidth = 2;
      ctx.strokeStyle = 'black';
      ctx.moveTo(
        this.rays[i][1].x,
        this.rays[i][1].y,
      );
      ctx.lineTo(
        endPoint.x,
        endPoint.y,
      );
      ctx.stroke();
    }
  }
}
```

Додаток Д. Скрипт нейронної мережі

```
import { NetworkLayer } from './network-layer';
import { getBrainFromStorage, lerp } from './utils';

export class NeuralNetwork {
  /**
   * A Neural Network to train on cars.
   * @param {number[]} neurons An array of neuron amounts for each level.
   */
  constructor(neurons) {
    this.layers = [];

    for (let i = 0; i < neurons.length - 1; i++) {
      this.layers.push(new NetworkLayer(neurons[i], neurons[i + 1]));
    }
  }

  /**
   * Use the Feed Forward algorithm to train the neural network.
   * @param {number[]} inputs
   * @param {NeuralNetwork} network
   */
  static feedForward(inputs, network) {
    let outputs = NetworkLayer.feedForward(inputs, network.layers[0]);

    for (let i = 1; i < network.layers.length; i++) {
      outputs = NetworkLayer.feedForward(outputs, network.layers[i]);
    }
    return outputs;
  }

  /**
   * Train the neural network further, by some amount (%).
   * @param {NeuralNetwork} network
   */
  static train(network, amount = 1) {
    network.layers.forEach((layer) => {
      for (let i = 0; i < layer.biases.length; i++) {
        layer.biases[i] = lerp(
          layer.biases[i],
          Math.random() * 2 - 1,
          amount,
        );
      }
      for (let i = 0; i < layer.weights.length; i++) {
        for (let j = 0; j < layer.weights[i].length; j++) {
          layer.weights[i][j] = lerp(
            layer.weights[i][j],
            Math.random() * 2 - 1,
            amount,
          );
        }
      }
    });
  }
}

/**
 * Get the stored best brain from the local storage, and train other cars.
 * @param {Car[]} cars
 * @param {number} trainValue
 */
```

```
export function trainCars(cars, trainValue = 0.1) {
  if (getBrainFromStorage()) {
    for (let i = 0; i < cars.length; i++) {
      cars[i].brain = getBrainFromStorage();

      if (i !== 0) {
        NeuralNetwork.train(cars[i].brain, trainValue);
      }
    }
  }
}
```

Додаток Е. Скрипт роботи з рівнями мережі

```
export class NetworkLayer {
  /**
   * A layer of a Neural Network.
   * @param {number} inputNeuronsCount
   * @param {number} outputNeuronsCount
   */
  constructor(inputNeuronsCount, outputNeuronsCount) {
    this.inputs = new Array(inputNeuronsCount);
    this.outputs = new Array(outputNeuronsCount);
    this.biases = new Array(outputNeuronsCount);

    this.weights = [];
    for (let i = 0; i < inputNeuronsCount; i++) {
      this.weights[i] = new Array(outputNeuronsCount);
    }

    this.#randomize();
  }

  #randomize() {
    for (let i = 0; i < this.inputs.length; i++) {
      for (let j = 0; j < this.outputs.length; j++) {
        this.weights[i][j] = Math.random() * 2 - 1;
      }
    }
    for (let i = 0; i < this.biases.length; i++) {
      this.biases[i] = Math.random() * 2 - 1;
    }
  }

  /**
   * Use the Feed Forward algorithm to train the neural network.
   * @param {number[]} inputs
   * @param {NetworkLayer} layer
   * @return {number[]} outputs
   */
  static feedForward(inputs, layer) {
    for (let i = 0; i < layer.inputs.length; i++) {
      layer.inputs[i] = inputs[i];
    }

    for (let i = 0; i < layer.outputs.length; i++) {
      let sum = 0;

      for (let j = 0; j < layer.inputs.length; j++) {
        sum += layer.inputs[j] * layer.weights[j][i];
      }
      if (sum > layer.biases[i]) {
        layer.outputs[i] = 1;
      } else {
        layer.outputs[i] = 0;
      }
    }
    return layer.outputs;
  }
}
```

Додаток Ж. Скрипт візуалізації мережі

```
import { NeuralNetwork } from './network';
import { NetworkLayer } from './network-layer';
import { getNetworkVisualizerColorctx, lerp } from './utils';

export class NetworkVisualizer {
  /**
   * Visualize the neural network on a canvas context.
   * @param {CanvasRenderingContext2D} ctx
   * @param {NeuralNetwork} network
   */
  static visualize(ctx, network) {
    const margin = 50;
    const left = margin;
    const top = margin;
    const width = ctx.canvas.width - margin * 2;
    const height = ctx.canvas.height - margin * 2;

    const levelHeight = height / network.layers.length;

    for (let i = network.layers.length - 1; i >= 0; i--) {
      const levelTop = top + lerp(
        height - levelHeight,
        0,
        network.layers.length === 1
          ? 0.5
          : i / (network.layers.length - 1),
      );

      ctx.setLineDash([7, 3]);

      NetworkVisualizer.#displayLevel(
        ctx,
        network.layers[i],
        left,
        levelTop,
        width,
        levelHeight,
        i === network.layers.length - 1
          ? ['□', '□', '□', '□']
          : [],
      );
    }
  }

  /**
   * @param {CanvasRenderingContext2D} ctx
   * @param {NetworkLayer} level
   * @param {number} left
   * @param {number} top
   * @param {number} width
   * @param {number} height
   * @param {string[]} outputLabels
   */
  static #displayLevel(ctx, level, left, top, width, height, outputLabels) {
    const right = left + width;
    const bottom = top + height;
    const nodeRadius = 18;
    const bgColor = 'black';

    const {
      inputs, outputs, weights, biases,
```

```
    } = level;

    for (let i = 0; i < inputs.length; i++) {
      for (let j = 0; j < outputs.length; j++) {
        ctx.beginPath();
        ctx.moveTo(NetworkVisualizer.#getNodeX(inputs, i, left, right),
bottom);
        ctx.lineTo(NetworkVisualizer.#getNodeX(outputs, j, left, right),
top);
        ctx.lineWidth = 2;
        ctx.strokeStyle = getNetworkVisualizerColorctx(weights[i][j]);
        ctx.stroke();
      }
    }

    for (let i = 0; i < inputs.length; i++) {
      const x = NetworkVisualizer.#getNodeX(inputs, i, left, right);

      ctx.beginPath();
      ctx.arc(x, bottom, nodeRadius, 0, Math.PI * 2);
      ctx.fillStyle = bgColor;
      ctx.fill();
      ctx.beginPath();
      ctx.arc(x, bottom, nodeRadius * 0.6, 0, Math.PI * 2);
      ctx.fillStyle = getNetworkVisualizerColorctx(inputs[i]);
      ctx.fill();
    }

    for (let i = 0; i < outputs.length; i++) {
      const x = NetworkVisualizer.#getNodeX(outputs, i, left, right);

      ctx.beginPath();
      ctx.arc(x, top, nodeRadius, 0, Math.PI * 2);
      ctx.fillStyle = bgColor;
      ctx.fill();
      ctx.beginPath();
      ctx.arc(x, top, nodeRadius * 0.6, 0, Math.PI * 2);
      ctx.fillStyle = getNetworkVisualizerColorctx(outputs[i]);
      ctx.fill();

      ctx.beginPath();
      ctx.lineWidth = 2;
      ctx.arc(x, top, nodeRadius * 0.8, 0, Math.PI * 2);
      ctx.strokeStyle = getNetworkVisualizerColorctx(biases[i]);
      ctx.setLineDash([3, 3]);
      ctx.stroke();
      ctx.setLineDash([]);

      if (outputLabels[i]) {
        ctx.beginPath();
        ctx.textAlign = 'center';
        ctx.textBaseline = 'middle';
        ctx.fillStyle = bgColor;
        ctx.strokeStyle = 'grey';
        ctx.font = `${nodeRadius * 1.5}px Arial`;
        ctx.fillText(outputLabels[i], x, top + nodeRadius * 0.1);
        ctx.lineWidth = 0.5;
        ctx.strokeText(outputLabels[i], x, top + nodeRadius * 0.1);
      }
    }
  }

  static #getNodeX(nodes, index, left, right) {
```

```
    return lerp(
      left,
      right,
      nodes.length === 1
        ? 0.5
        : index / (nodes.length - 1),
    );
  }
}
```

Додаток К. Скрипт підключення контролю

```
export class Controls {
  /**
   * Creates a controlling mechanism for a car.
   * @param {'DUMMY' | 'AI'} type The type of a car.
   */
  constructor(type) {
    this.forward = false;
    this.left = false;
    this.right = false;
    this.backward = false;

    switch (type) {
      case 'AI': {
        this.#listenToKeyboardEvents();
        break;
      }
      case 'DUMMY': {
        this.forward = true;
        break;
      }
      default: {
        throw new Error('Wrong car type');
      }
    }
  }

  #listenToKeyboardEvents() {
    document.onkeydown = (event) => {
      switch (event.key) {
        case 'ArrowLeft': {
          this.left = true;
          break;
        }
        case 'ArrowRight': {
          this.right = true;
          break;
        }
        case 'ArrowUp': {
          this.forward = true;
          break;
        }
        case 'ArrowDown': {
          this.backward = true;
          break;
        }
        default: {
          break;
        }
      }
    };
    document.onkeyup = (event) => {
      switch (event.key) {
        case 'ArrowLeft': {
          this.left = false;
          break;
        }
        case 'ArrowRight': {
          this.right = false;
          break;
        }
        case 'ArrowUp': {
```

```
            this.forward = false;
            break;
          }
          case 'ArrowDown': {
            this.backward = false;
            break;
          }
          default: {
            break;
          }
        }
      };
    }
  }
```

Додаток Л. Скрипт із додатковими функціями

```
/**
 * The Linear Interpolation formula.
 * @param {number} a
 * @param {number} b
 * @param {number} t
 */
export function lerp(a, b, t) {
  return a + (b - a) * t;
}

/**
 * Get an intersection of two objects.
 * @param {{x: number, y: number}} a
 * @param {{x: number, y: number}} b
 * @param {{x: number, y: number}} c
 * @param {{x: number, y: number}} d
 * @return {{x: number, y: number, offset: number} | null} The intersection
point.
 */
export function getIntersection(a, b, c, d) {
  const tNumerator = (d.x - c.x) * (a.y - c.y) - (d.y - c.y) * (a.x - c.x);
  const uNumerator = (c.y - a.y) * (a.x - b.x) - (c.x - a.x) * (a.y - b.y);
  const denominator = (d.y - c.y) * (b.x - a.x) - (d.x - c.x) * (b.y - a.y);

  if (denominator) {
    const t = tNumerator / denominator;
    const u = uNumerator / denominator;

    if (t >= 0 && t <= 1 && u >= 0 && u <= 1) {
      return {
        x: lerp(a.x, b.x, t),
        y: lerp(a.y, b.y, t),
        offset: t,
      };
    }
  }
  return null;
}

/**
 *
 * @param {{x: number, y: number}[]} polygon1
 * @param {{x: number, y: number}[]} polygon2
 * @returns
 */
export function isPolygonIntersected(polygon1, polygon2) {
  for (let i = 0; i < polygon1.length; i++) {
    for (let j = 0; j < polygon2.length; j++) {
      const isTouched = getIntersection(
        polygon1[i],
        polygon1[(i + 1) % polygon1.length],
        polygon2[j],
        polygon2[(j + 1) % polygon2.length],
      );
      if (isTouched) {
        return true;
      }
    }
  }
  return false;
}
```

```
export function getNetworkVisualizerColor(value) {
  const alpha = Math.abs(value);
  return 'rgba(' + (value < 0 ? '255, 76, 48, ' : '147, 250, 165, ') +
`${alpha + 0.2})`;
}

export function getRGBA(value) {
  const alpha = Math.abs(value);
  const R = value < 0 ? 0 : 255;
  const G = R;
  const B = value > 0 ? 0 : 255;
  return `rgba(${R}, ${G}, ${B}, ${alpha})`;
}

export function getRandomColor() {
  const hue = getRandomNum(290, 290 + 260);
  return `hsl(${hue}, 100%, 60%)`;
}

export function getRandomNum(start = 0, finish = 1) {
  return Math.floor(Math.random() * finish + start);
}

export const brainLabel = 'bestBrain';

/**
 * Get the stored best brain from the local storage, or nothing...
 * @return {{
 * levels: {inputs: number[], outputs: number[], biases: number[], weights:
number[]}[]
 * } | undefined}
 */
export function getBrainFromStorage() {
  return getLocalStorageItem(brainLabel);
}

/**
 * Store the best brain to the local storage.
 * @param {{
 * levels: {inputs: number[], outputs: number[], biases: number[], weights:
number[]}[]
 * }} brain
 */
export function saveBrainToStorage(brain) {
  saveToLocalStorageItem(brainLabel, brain);
}

export function removeBrainFromStorage() {
  removeItemFromLocalStorage(brainLabel);
}

/**
 * Get an item from the local storage.
 * @param {string} item
 */
export function getLocalStorageItem(item) {
  return JSON.parse(localStorage.getItem(item));
}

/**
 * Save an item to the local storage.
 * @param {string} key
 * @param {string} value
```

```
 */
export function saveToLocalStorageItem(key, value) {
  localStorage.setItem(key, JSON.stringify(value));
}

/**
 * Remove an item from the local storage.
 * @param {string} key
 */
export function removeItemFromLocalStorage(key) {
  localStorage.removeItem(key);
}
```