**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ**

**Чернівецький національний університет імені Юрія Федьковича**

**Кафедра комп’ютерних наук**

**КУРСОВА РОБОТА**

на тему:

"Визначення жанру комп'ютерної гри за її ознаками"

Виконав студент

групи 344 \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Д.Р. Скібінський

(підпис)

Науковий керівник \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ доцент М.Л. Ковальчук

(підпис)

Національна шкала \_\_\_\_\_\_\_ Кількість балів \_\_\_\_\_\_\_ Оцінка ECTS \_\_\_\_\_

Чернівці – 2021

[Вступ 3](#_Toc72810068)

[Розділ 1. Теоретичні відомості про нейронні мережі 5](#_Toc72810069)

[1.1 Нейрон 5](#_Toc72810070)

[1.2 Синапс 5](#_Toc72810071)

[1.3 Активаційна функція 6](#_Toc72810072)

[1.4 Ітерація та епоха 7](#_Toc72810073)

[1.5 Помилка 8](#_Toc72810074)

[1.6 Нейрон зміщення 8](#_Toc72810075)

[1.7 Гіперпараметри 9](#_Toc72810076)

[1.8 Типи навчання мереж 9](#_Toc72810077)

[1.9 Навчання нейромережі 10](#_Toc72810078)

[1.10 Метод зворотного поширення 10](#_Toc72810079)

[1.11 Градієнтний спуск 10](#_Toc72810080)

[Розділ 2. Опис предметної області та використаних технологій 13](#_Toc72810081)

[2.1 Опис використаних технологій 13](#_Toc72810082)

[2.2 Опис бази даних 13](#_Toc72810083)

[2.3 Опис XML файлів для збереження вагових коефіцієнтів 14](#_Toc72810084)

[2.4 Опис роботи нейронної мережі в додатку 15](#_Toc72810085)

[2.5 Опис функціональних вимог додатку 17](#_Toc72810086)

[Розділ 3. Опис створеного програмного продукту 19](#_Toc72810087)

[3.1 Архітектура додатку 19](#_Toc72810088)

[3.2 DatabaseContext 20](#_Toc72810089)

[3.3 Controllers 20](#_Toc72810090)

[3.4 NeuronNetwork 21](#_Toc72810091)

[3.5 UI 22](#_Toc72810092)

[Висновок 29](#_Toc72810093)

[Список використаної літератури 30](#_Toc72810094)

[Додаток А 31](#_Toc72810095)

[Додаток A (Продовження) 32](#_Toc72810096)

[Додаток Б 33](#_Toc72810097)

[Додаток В 34](#_Toc72810098)

[Додаток Г 35](#_Toc72810099)

[Додаток Д 35](#_Toc72810100)

[Додаток Е 36](#_Toc72810101)

# Вступ

Нині ми живемо в той час, коли все більше і більше процесів в житті людини автоматизуються і численні проблеми вирішуються за допомогою комп’ютера. В останні роки особливого поширення набули штучний інтелект та нейронні мережі. Структура нейронних мереж бере свій початок з біології, а прототипом штучного нейрона є однойменна клітина людського організму. Однією з найцікавіших властивостей нейронної мережі є те, що вона здатна не лише аналізувати вхідні дані, а і відтворювати їх на основі проаналізованої інформації. Ця можливість аналітичних обчислень дає змогу використовувати нейронні мережі в ряді задач, зокрема в задачах класифікації, передбачення та розпізнавання.

Одна з таких задач буде опрацьована в цій курсовій роботі, а саме задача розпізнавання та класифікації відеоігор на основі набору певних їх ознак та віднесення її до певного жанру. Ця задача є типовою для розв’язання нейронною мережею, оскільки значна частина ознак тої чи іншої гри та різноманіття жанрів робить задачу ручної класифікації ігор достатньо трудомісткою, а використовуючи вище згадану властивість можна навчити нейронну мережу на відносно невеликому наборі вхідних даних.

Об’єктом дослідження курсової роботи є власне нейронна мережа та нейрон як такий.

Предметом вивчення курсової роботи є дослідження алгоритмів роботи та навчання нейронних мереж, а також їх реалізація на прикладі конкретного завдання класифікації ігор.

Метою написання курсової роботи є створення програмного продукту, який реалізує один з алгоритмів навчання нейронної мережі.

Було вирішено реалізувати нейронну мережу у вигляді окремого веб додатку, написаного на мові програмування C#, який надає можливість користувачу зручно заповнити інформацію про існуючі жанри та ознаки ігор, навчити нейронну мережу на основі конкретних прикладів та перевірити здатність нейронної мережі класифікувати гру з невідомими наборами ознак.

Курсова робота складається з трьох наступних розділів: теоретичні відомості та основні поняття нейронних мереж, опис предметної області та технологій, опис створеного програмного продукту.

В першому розділі буде розглянуто основні питання, що стосуються створення, навчання та використання нейронної мережі.

В другому розділі буде описано предметну область та набір технологій для реалізації нейронної мережі.

В третьому розділі буде показано роботу мережі на прикладі розробленого веб додатку, описано його структуру та інтерфейс.

# Розділ 1. Теоретичні відомості про нейронні мережі

## 

## 1.1 Нейрон

Нейрон - це обчислювальна одиниця в нейронній мережі. Розрізняють три основних типа нейрону: вхідний, схований та вихідний.

При значній кількості нейронів вводять поняття шару нейронів. Зазвичай в нейронній мережі є вхідний шар, декілька схованих шарів (зазвичай не більше 3) та вихідний шар.

Кожен нейрон містить наступні параметри: набір вхідних сигналів та вихідний сигнал. Для вхідних нейронів вихідний сигнал рівний вхідному, в усі інші нейрони потрапляє сумарна інформація з нейронів попередніх шарів, яка потім нормалізується за допомогою функції активації.

## 1.2 Синапс

Синапс - це зв’язок між двома нейронами. Кожен синапс має певний ваговий коефіцієнт (Рисунок 1). Саме на основі цих зв’язків формується вихідний сигнал нейрона. Чим більший ваговий коефіцієнт синапсу, тим більше впливає сигнал з попереднього нейрона на наступний.

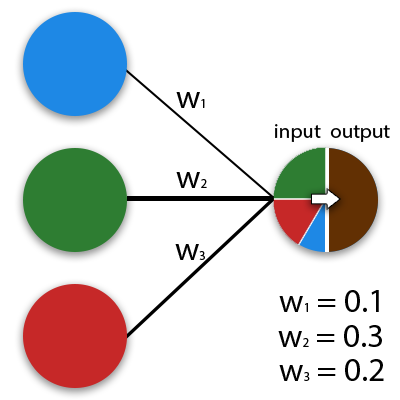


Рисунок 1. Синапси

## 1.3 Активаційна функція

Вихідний сигнал нейрона рахується шляхом сумування добутків вхідних сигналів на відповідні вагові коефіцієнти, після чого отримана сума нормалізується активаційною функцією. Основними активаційними функціями є лінійна функція (рисунок 2), сигмоїда (рисунок 3), гіперболічний тангенс (рисунок 4).

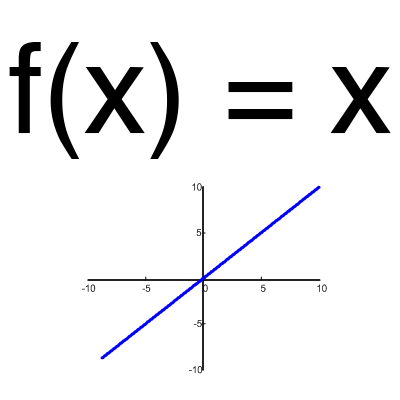


Рисунок 2. Лінійна функція

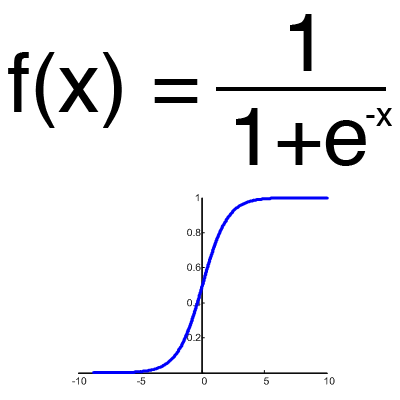


Рисунок 3. Сигмоїда

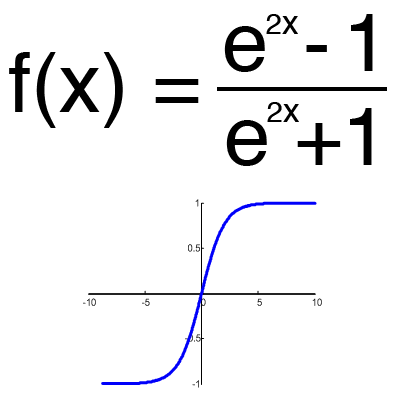


Рисунок 4. Гіперболічний тангенс

## 1.4 Ітерація та епоха

Нейронна мережа тренується на так званих тренувальних наборах - сукупності вхідних та вихідних значень нейронної мережі. Проходження кожного тренувального набору називається ітерацією. Проходження усіх тренувальних наборів по одному разу називається епохою. Важливо пам’ятати, що неможлива натренувати нейронну мережу на кожному з наборів окремо. За одну епоху нейронна мережа має пройти по кожному з тренувальних наборів по разу.

## 1.5 Помилка

Помилка - це процентна величина, яка вказує на розбіжність між отриманим результатом та бажаним. Помилка має формуватися один раз за епоху та з кожною новою епохою йти на спад. Існує кілька методів пошуку помилки. Найпоширенішим з них є метод MSE (Mean Squared Error), зображений на рисунку 5.

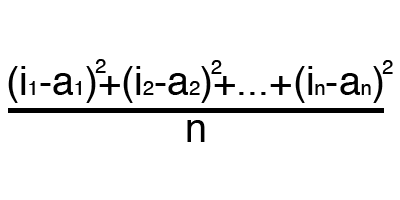


Рисунок 5. MSE

## 1.6 Нейрон зміщення

Нейрон зміщення - особливий нейрон, який використовується в нейромережах. Його особливість полягає в тому, що його вхід та вихід завжди рівні 1, він ніколи не має вхідних синапсів. Він може використовуватись для зміщення графіку функції по вісі абсцис для знаходження необхідного результату. Ілюстрація цього зображена на рисунку 6.

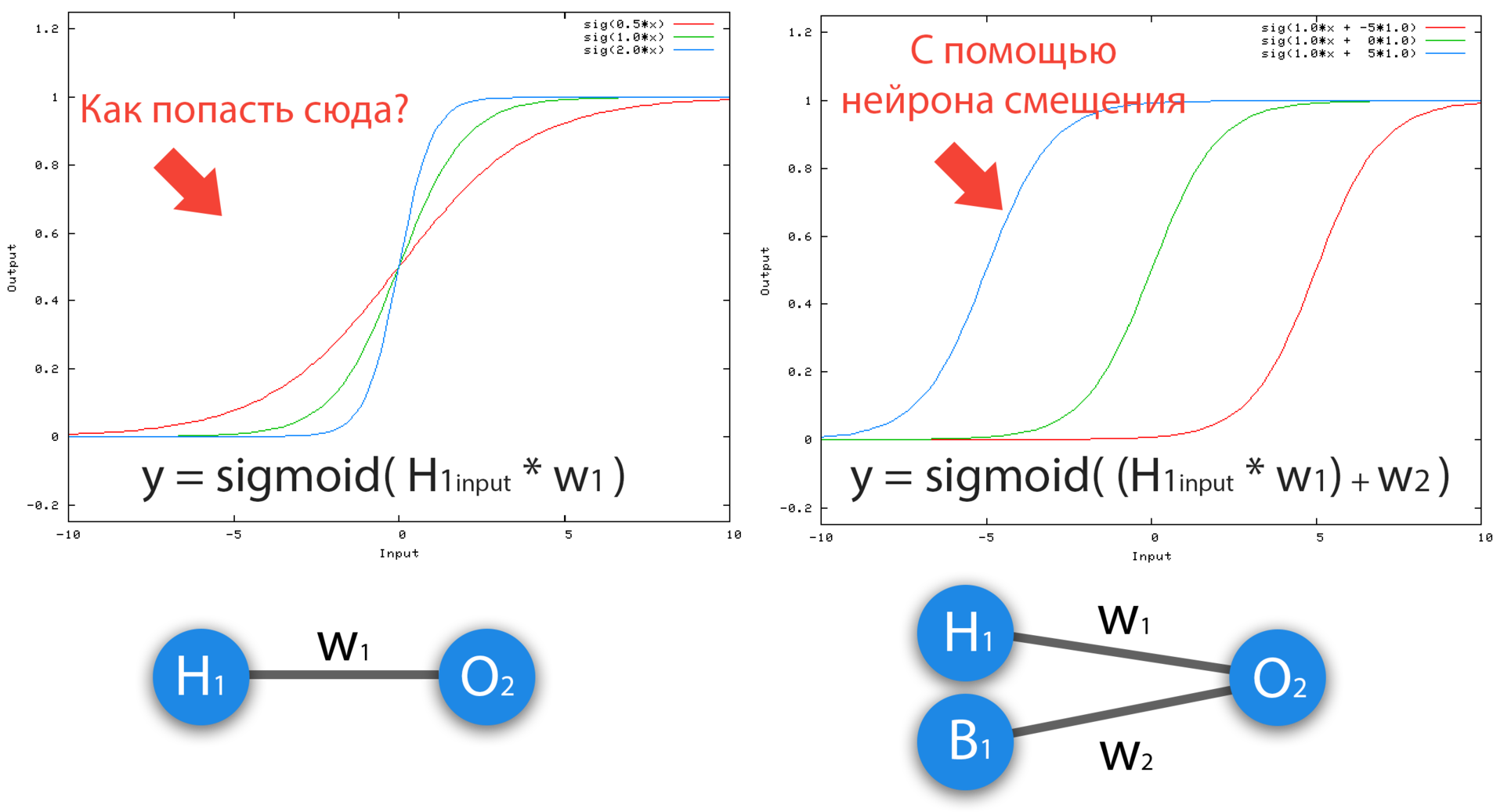


Рисунок 6. Нейронне зміщення

## 1.7 Гіперпараметри

Гіперпараметри - це набір значень в нейромережі, які зазвичай підбираються вручну. Серед таких параметрів виділяють наступні:

* Момент та швидкість навчання
* Кількість прихованих шарів
* Кількість нейронів в кожному шарі
* Присутність або відсутність нейронів зміщення.

## 1.8 Типи навчання мереж

Існують наступні типи навчання нейромережі: навчання з вчителем, навчання без вчителя, навчання з підкріпленням.

Навчання з вчителем – це тип навчання, в якому людина виступає в ролі вчителя, а нейромережа в ролі учня. Вчитель надає вхідні дані і бажаний результат, а учень, використавши вхідні дані, зрозуміє, що потрібно прагнути до бажаного результату.

Навчання без вчителя – це тип навчання, при якому вчитель відсутній і нейромережа не отримує бажаного результату. Такий тип навчання характерний для нейромереж, для яких задача полягає в групуванні даних за певними параметрами.

Навчання з підкріпленням – це тип навчання, який застосовується тоді, коли ми можем дати певну оцінку нейромережі, засновану на результатах її роботи. В залежності від цієї оцінки нейромережа буде коригувати свою роботу і буде намагатись знайти найкращий результат досягнення мети без постійного отримання даних від вчителя.

## 1.9 Навчання нейромережі

Після першого проходження нейронної мережі необхідно застосувати один з методів корекції вагових коефіцієнтів. Найпоширенішими методами навчання нейромережі є наступні:

* Метод зворотного поширення
* Метод еластичного зворотного поширення
* Генетичний алгоритм

## 1.10 Метод зворотного поширення

Спосіб проходу нейронної мережі від вхідних нейронів до вихідних називається проходом вперед. Після проходу всієї нейромережі необхідно знайти помилку, і на основі цієї помилки необхідно послідовно змінити усі вагові коефіцієнти нейронів, починаючи з вихідних. Значення нейронів буде змінюватись в кращу сторону, щоб досягти найкращого результату. Цей метод несе назву Метод зворотнього поширення.

## 1.11 Градієнтний спуск

Градієнтним спуском називають спосіб знаходження локального мінімум чи максимуму функції за допомогою переміщення вздовж градієнта. На рисунку 7 зображено графік нашої функції, де на вісі абсцис знаходиться значення вагового коефіцієнта нейрона, а на вісі ординат – значення помилки, що відповідає даному ваговому коефіцієнту.

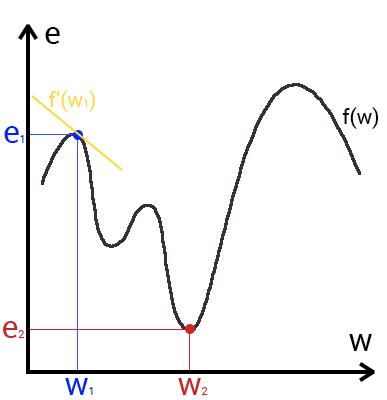


Рисунок 7. Графік функції вагового коефіцієнта

Даний графік зображує функцію f(w), яка виражає залежність помилки від вибраного вагового коефіцієнта. На цьому графіку нас цікавить глобальний мінімум – точка (w2, e2), тобто місце, де графік найближче наближається до вісі абсцис. Ця точка означає, що обравши ваговий коефіцієнт w2 ми отримаємо найменшу помилку – e2, а отже і найкращий результат з усіх можливих. Градієнт на графіку позначено жовтим кольором. Для кожного вагового коефіцієнта в нейромережі буде існувати свій графік і градієнт і для кожного необхідно буде знайти глобальний мінімум.

Градієнт – це вектор, який визначає нахил графіку і вказує його напрямок відносно якої-небудь з точок на поверхності чи графіку. Для того, щоб визначити градієнт необхідно взяти похідну від графіка в даній точці (як і показано на рисунку). Рухаючись в напрямку цього градієнта ми будем поступово рухатись в сторону мінімуму. На наступному рисунку зображено вигляд градієнтного спуску, де червоним позначено шлях, по якому ми будемо рухатись вздовж градієнту.

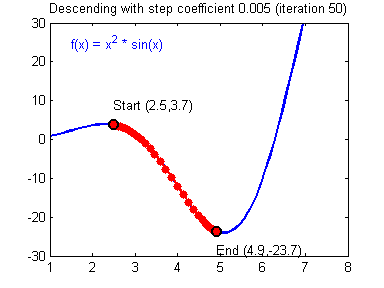


Рисунок 8. Градієнтний спуск

# Розділ 2. Опис предметної області та використаних технологій

## 2.1 Опис використаних технологій

Для створення веб додатку було використано фреймворк ASP .NET Core MVC, який реалізує шаблон Model-View-Controller для створення backend та frontend частин додатку. Для розробки було використано середовище програмування Microsoft Visual Studio 2019 Community та мову C# 9 версії (платформа .NET 5).

Було також вирішено організувати роботу з базою даних MS SQL Server, для цього було використано вбудований в Microsoft Visual Studio плагін SQL Object Explorer. Для операцій зчитування та запису було використано Entity Framework Core, а запити до бази даних написані за допогою технології LINQ to Entities.

Користувацький інтерфейс було написано з використанням технологій HTML, CSS, Razor Pages та Bootstrap.

Дані про вагові коефіцієнти нейронної мережі було записано в окремі XML файли.

## 2.2 Опис бази даних

Було створено декілька таблиць в базі даних MS SQL Server з використанням Entity Framework та підходу Code First.

Для цього було виділено наступні чотири сутності:

* GameAspects (Id, Name) - основні ознаки (аспекти) досліджуваних ігор, які беруться за вхідні нейрони
* GameGenres (Id, Name) - жанри досліджуваних ігор, які беруться за вихідні нейрони
* TrainingSets(Id, GameGenreId, GameAspectGroupId) - тренувальні вибірки
* GameAspectGroups(GroupId, GameAspectId, Value) - зберігає інформацію про значення вхідних сигналів для конкретної вибірки

На рисунку 9 зображено діаграму створеної бази даних.

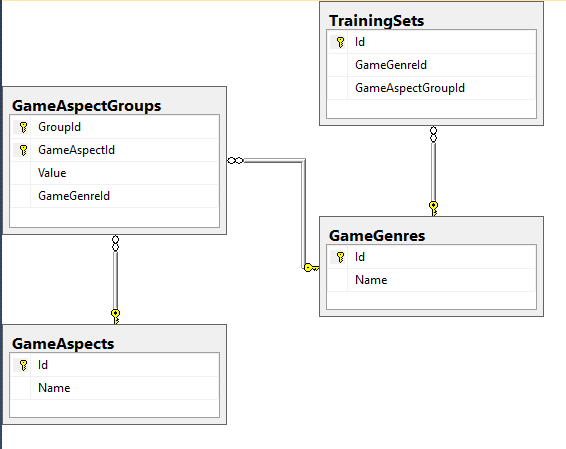


Рисунок 9. База даних додатку

## 2.3 Опис XML файлів для збереження вагових коефіцієнтів

Вагові коефіцієнти, знайдені після навчання нейромережі зберігаються для подальшого використання в двох файлах XML, а саме hidden\_layer\_memory.xml з ваговими коефіцієнтами прихованого рівню та output\_layer\_memory.xml з ваговими коефіцієнтами вихідного рівню. Варто зазначити, що збережені вагові коефіцієнти для кожного шару є ваговими коефіцієнтами для зв’язку з нейронами попереднього шару.

Структура створених XML файлів зображена на рисунку 10.

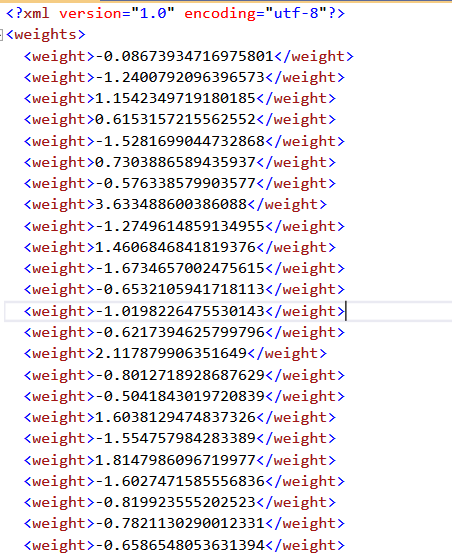


Рисунок 10. Структура створеного XML файлу

Як видно з рисунку, було створено кореневий елемент <weights>, який вміщає в собі дочірні елементи <weight>, кожен з яких містить ваговий коефіцієнт нейромережі.

Окремо слід відзначити файл clear\_layer\_memory.xml. Він зберігає початковий вигляд xml файлів і використовується для перенавчання нейронної мережі.

## 2.4 Опис роботи нейронної мережі в додатку

Нейромережа для створеного додатку була написана з нуля на чистій мові C# без використання жодних сторонніх бібліотек задля навчальних цілей. Структура мережі була створена з використанням основних принципів ООП (абстракція, інкапсуляцій, наслідування та поліморфізм) та принципів SOLID.

Створена нейромережа складається з декількох шарів, кожен з яких містить певний набір нейронів та вагові коефіцієнти для зв’язку з нейронами попередньго шару. Нейрони в цих шарах об’єднуються в масиви а не в списки для більшої продуктивності на великих об’ємах даних. Оскільки основна логіка роботи шарів нейромережі є подібною, то було вирішено створити абстрактний клас з базовим для всіх шарів функціоналом та використати наслідування для передачі цього функціоналу дочірнім класам.

Кількість нейронів на вхідному та вихідному шарах нейромережі є динамічною і дорівнює кількості ознак відеоігор та кількості записаних в базу даних жанрів відповідно. Нейромережа містить один прихований шар, кількість нейронів в прихованому шарі є константою, яку можна змінити до запуску додатку.

Серед констант також знаходяться інші гіперпараметри, такі як швидкість навчання та точність обчислень. Нейрон зміщення в мережі не використовується.

Активаційною функцією була обрана сигмоїда, а пошук помилки здійснено методом MSE.

Після навчання до шуканого рівня точності вагові коефіцієнти мережі записуються в окремі XML файли і при роботі мережі зчитуються з цих файлів. Навчання мережі здійснюється методом зворотного поширення.

Спрощена схема нейронної мережі зображена на рисунку 11.

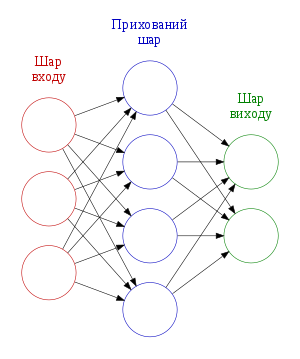


Рисунок 11. Схема нейромережі

## 2.5 Опис функціональних вимог додатку

До додатку було висунуто наступні вимоги:

* Веб-додаток зі зручним та зрозумілим інтерфейсом
* Можливість переглядати, додавати, видаляти та редагувати основні ознаки відеоігор (вхідний набір даних)
* Можливість переглядати, додавати, видаляти та редагувати жанри відеоігор (бажаний вихідний набір даних)
* Можливість переглядати та створювати тренувальні набори даних для навчання нейромережі на основі вже створених вхідних та вихідних даних з динамічним завантаженням даних в випадаючі списки та прапорці.
* Можливість навчати нейронну мережу на основі раніше створених тренувальних наборів
* Можливість зберегти знайдені вагові коефіцієнти мережі після процесу навчання
* Можливість використати нейронну мережу зі збереженими ваговими коефіцієнтами для вгадування жанру комп’ютерної гри за вказаними ознаками

# Розділ 3. Опис створеного програмного продукту

## 3.1 Архітектура додатку

Як вже зазначалось вище, для створеного додатку було використано монолітну архітектуру, що означає, що було створено один проект, який вміщує в собі всю логіку роботи. На рисунку 7 зображено структуру створеного проекту.

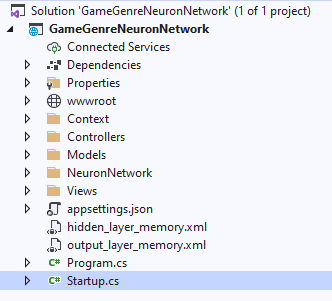


Рисунок 12. Структура проекту

Як видно з рисунку - проект створено згідно шаблону MVC. Це означає, що додаток містить три основні аспекти: Models - сукупність “чистих” моделей додатку, які не містять жодної логіки, Views - сукупність сторінок додатку, які також практично не містять логіки і відповідають лише за відображення даних, та Controllers - сукупність контролерів, в яких зосереджена вся логіка (бізнес логіка, логіка роботи з БД). Серед шарфів варто ще відмітити шар Context, який містить EF DbContext, через який проходять всі операції бази даних та який відповідає за сідінг даних, а також NeuronNetwork, в якому міститься логіка роботи нейронної мережі.

## 3.2 DatabaseContext

Клас DatabaseContext, код якого наведено нижче, відповідає за роботи з базою даних. В ньому вміщаються класи DbSet, які відповідають таблицям з бази даних. Строка підключення до бази даних прописана у файлі AppSettings.json, а початкова конфігурація в файлі Startup.cs.

public class DatabaseContext : DbContext

{

public DbSet<GameAspect> GameAspects { get; set; }

public DbSet<GameGenre> GameGenres{ get; set; }

public DbSet<GameAspectGroup> GameAspectGroups { get; set; }

public DbSet<TrainingSet> TrainingSets { get; set; }

…

}

## 3.3 Controllers

Контроллеры - це своєрідні backend сервіси проекту. В них через Depndency Injection викликається DbContext і відбувається зчитування даних з та записування даних в базу даних. Для цього контролери оперують сутностями, створеними в шарі Models. Крім того, контролери повертають потрібні веб-сторінки на запит користувача, створені в шарі View. Одни з основних контролерів є LearningController, код якого приведено нижче. Саме він починає навчання нейронної мережі.

public class LearningController : Controller

{

private readonly DatabaseContext \_context;

public LearningController(DatabaseContext context)

{

\_context = context;

}

public IActionResult Index()

{

return View();

}

[HttpPost]

[ValidateAntiForgeryToken]

public async Task<IActionResult> Learn()

{

Network net = new Network(\_context);

Network.Train(net);

return RedirectToAction("Index", "Home");

}

}

Код запитів до бази даних написаний на мові запитів LINQ, синтаксис якої нагадує мову SQL (рисунок 8).

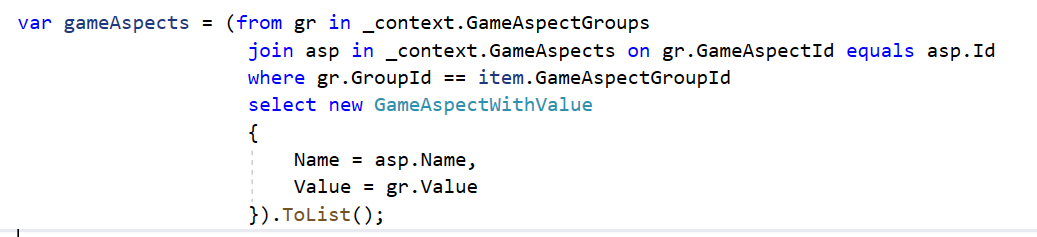


Рисунок 13. Запит написаний на LINQ

## 3.4 NeuronNetwork

Основна логіка роботи з нейронною мережею описана в класах:

* Neuron.cs (логіка роботи нейрону, вагові коефіцієнти, активаційна функція)
* Network.cs (логіка навчання, пошуку похибки, тестування навченої мережі)
* HiddenLayer, InputLayer, OutputLayer, Layer (робота з шарами нейронної мережі)

Робота додатку з нейронною мережею організована за допомогою двох контролерів: LearningController та GuessingController, які містять методи для навчання та використання вже навченої нейронної мережі та викликають методи класу нейронної мережі Network – Train та Guess. Серед інших методів варто знаходяться методи GetMSE для пошуку помилки ітерації, GetCost для пошуку помилки епохи та конструктор з ініціалізацією шарів мережі.

Клас вхідного шару InputLayer містить логіку роботи з вхідними даними. Вхідні дані отримуються з раніше введених в базу даних тренувальних наборів через використання Entity Framework та LINQ to Entities.

Класи HiddenLayer та OutputLayer містять логіку прямого та зворотнього проходу нейронної мережі та наслідують абстрактний клас Layer. Клас Layer містить логіку роботи з XML файлами та ініціалізацією вагових коефіцієнтів.

Клас Neuron містить активаційну функцію, методи для обчислення градієнтів та похідних.

Код вище згаданих класів наведено в додатках.

## 3.5 UI

Користувацький інтерфейс програми представлений у вигляді декількох вебсторінок, на яких користувач може конфігурувати та тестувати роботу мережі. Ці вебсторінки були створені відповідно до функціональних вимог, описаних в другому розділі. Серед основних елементів інтерфейсу варто виділити наступні:



Рисунок 14. Навігаційна панель

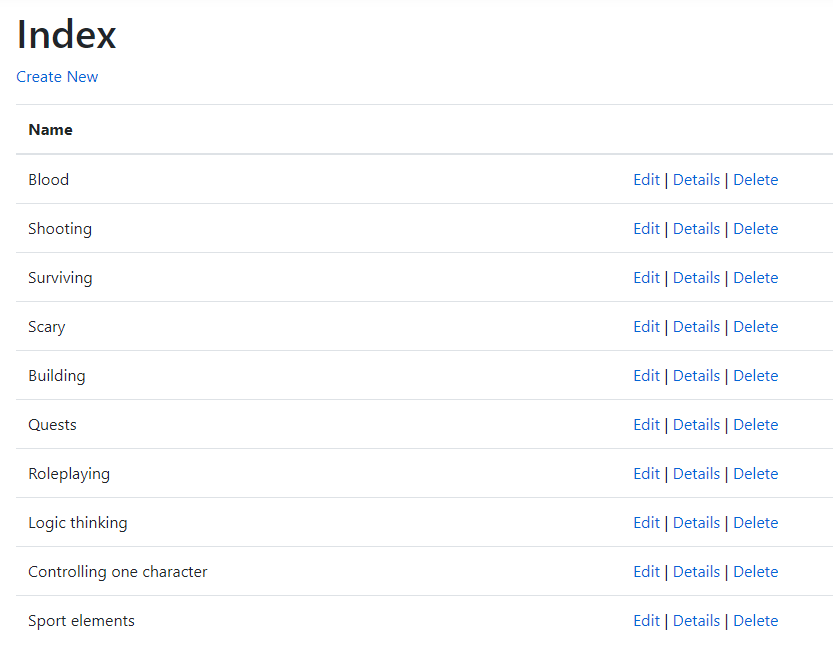


Рисунок 15. Сторінка роботи з основними ознаками ігор

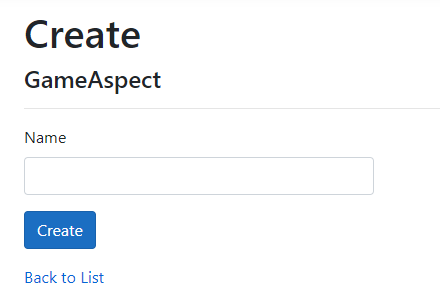


Рисунок 16. Форма створення ознаки гри

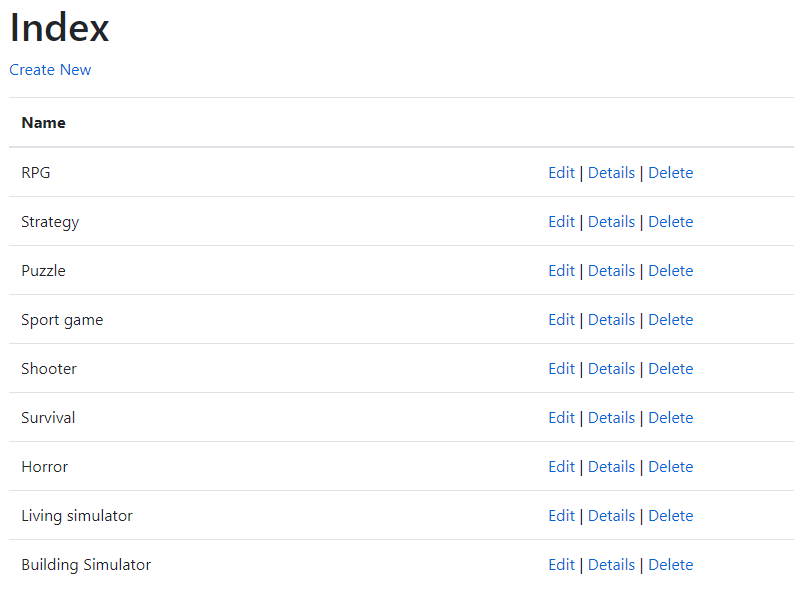


Рисунок 17. Сторінка роботи з жанрами ігор

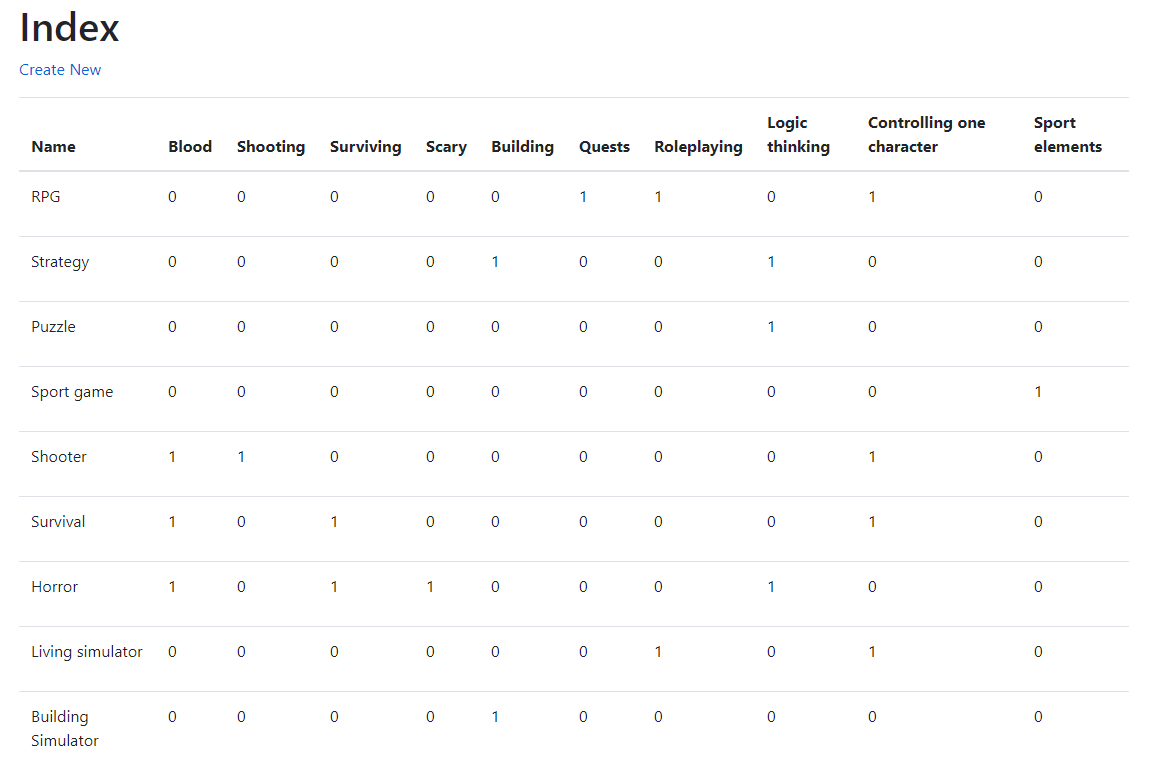


Рисунок 18. Таблиця роботи з навчальними вибірками

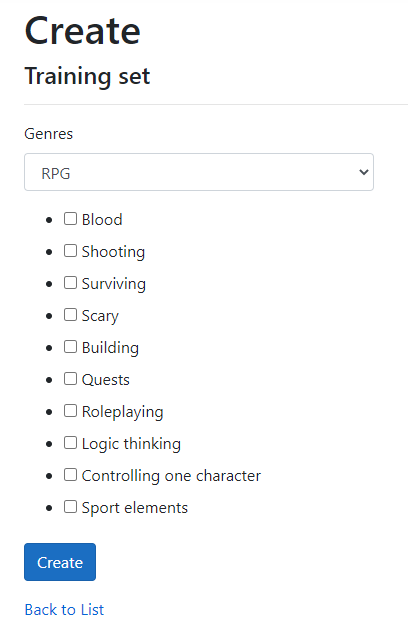


Рисунок 19. Форма створення навчальної вибірки

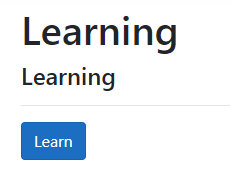


Рисунок 20. Кнопка навчання нейромережі

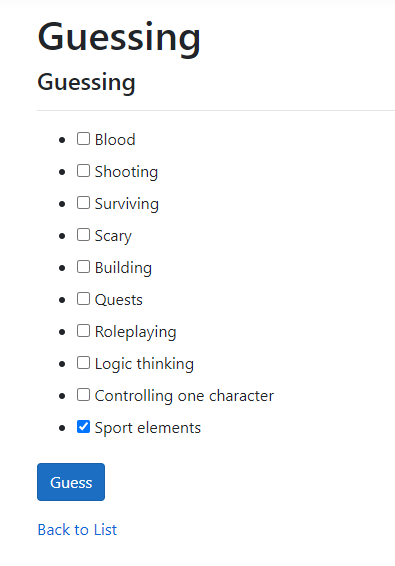


Рисунок 21. Форма тестування нейромережі

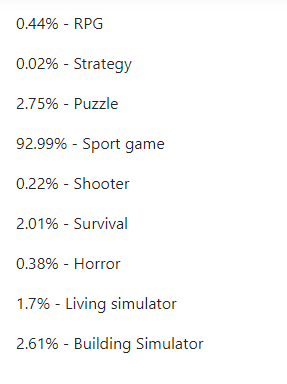


Рисунок 22. Вивід результатів нейромережі

# Висновок

В результаті виконання курсової роботи було досліджено принципи роботи нейронної мережі для виконання завдання класифікації на основі конкретного прикладу. Було створено веб додаток на платформі ASP .NET та мові програмування C#, який надає користувачеві змогу за допомогою зрозумілого інтерфейсу за швидкий час класифікувати необхідний набір даних.

Варто зазначити, що розроблений додаток можна використовувати не лише для класифікації відеоігор за жанрами на основі їх ознак, а й для класифікації будь-якого іншого набору даних з будь-якої іншої галузі людської діяльності, оскільки алгоритм навчання нейронної мережі є достатньо універсальним і є досить гнучким.

Створена нейронна мережа навчилася розпізнавати жанр відеогри за її ознаками, що дозволить зекономити час при класифікації ігор, наприклад людині, що заповнює інформацію про відеоігри в інтернет магазині чи на інформаційному порталі.

Аналогічно нейронні мережі можуть застосовуватись і в інших сферах людського життя для поліпшення продуктивності та економії часу.

# Список використаної літератури

1. Нейронные сети для начинающих. [Електронний ресурс] – Режим доступу до ресурсу: <https://habr.com/en/post/312450/>.
2. Neural Networks for Java [Електронний ресурс] – Режим доступу до ресурсу: <https://www.youtube.com/playlist?list=PLHIAnqK_uV8FJlIh04cUMZMPWNHryN7Ma>.
3. A Step by Step Backpropagation Example [Електронний ресурс] – Режим доступу до ресурсу: <https://mattmazur.com/2015/03/17/a-step-by-step-backpropagation-example/>.
4. Mind: How to Build a Neural Network [Електронний ресурс] – Режим доступу до ресурсу: <http://stevenmiller888.github.io/mind-how-to-build-a-neural-network/>.
5. ASP.NET Core MVC Введение в MVC [Електронний ресурс] – Режим доступу до ресурсу: <https://metanit.com/sharp/aspnet5/3.1.php>.
6. Руководство по Entity Framework Core [Електронний ресурс] – Режим доступу до ресурсу: https://metanit.com/sharp/entityframeworkcore/.

# Додаток А

**Код файлу Network.cs**

public class Network

{

private readonly DatabaseContext \_context;

public Network(DatabaseContext context)

{

input\_layer = new InputLayer(context);

int numberOfInputs = context.GameAspects.Count();

int numberOfOutputs = context.GameGenres.Count();

hidden\_layer = new HiddenLayer(Constants.NumberOfHiddenNeurons,

numberOfInputs,

NeuronType.Hidden,

nameof(hidden\_layer));

output\_layer = new OutputLayer(numberOfOutputs,

Constants.NumberOfHiddenNeurons,

NeuronType.Output,

nameof(output\_layer));

fact = new double[numberOfOutputs];

\_context = context;

}

InputLayer input\_layer;

public HiddenLayer hidden\_layer;

public OutputLayer output\_layer;

public double[] fact;

double GetMSE(double[] errors)

{

double sum = 0;

for (int i = 0; i < errors.Length; ++i)

sum += Math.Pow(errors[i], 2);

return 0.5d \* sum;

}

double GetCost(double[] mses)

{

double sum = 0;

for (int i = 0; i < mses.Length; ++i)

sum += mses[i];

return (sum / mses.Length);

}

public static void Train(Network net)

{

const double threshold = Constants.Accuracy;

double[] temp\_mses = new double[10];

# Додаток A (Продовження)

double temp\_cost;

do

{

for (int i = 0; i < net.input\_layer.Trainset.Length; ++i)

{

net.hidden\_layer.Data = net.input\_layer.Trainset[i].Item1;

net.hidden\_layer.Recognize(null, net.output\_layer);

net.output\_layer.Recognize(net, null);

double[] errors = new double[net.input\_layer.Trainset[i].Item2.Length];

for (int x = 0; x < errors.Length; ++x)

errors[x] = net.input\_layer.Trainset[i].Item2[x] - net.fact[x];

temp\_mses[i] = net.GetMSE(errors);

double[] temp\_gsums = net.output\_layer.BackwardPass(errors);

net.hidden\_layer.BackwardPass(temp\_gsums);

}

temp\_cost = net.GetCost(temp\_mses);

Console.WriteLine($"{temp\_cost}");

} while (temp\_cost > threshold);

net.hidden\_layer.WeightInitialize(MemoryMode.SET, nameof(hidden\_layer));

net.output\_layer.WeightInitialize(MemoryMode.SET, nameof(output\_layer));

}

public string[] Guess(Network net, double[] aspects)

{

net.hidden\_layer.Data = aspects;

net.hidden\_layer.Recognize(null, net.output\_layer);

net.output\_layer.Recognize(net, null);

var results = new List<string>();

var genres = \_context.GameGenres.ToArray();

for (int j = 0; j < net.fact.Length; ++j)

results.Add($"{Math.Round(net.fact[j] \* 100, 2)}% - {genres[j].Name}");

return results.ToArray();

}

}

# Додаток Б

**Код файлу Layer.cs**

public abstract class Layer {

protected Layer(int non, int nopn, NeuronType nt, string type) {

numofneurons = non;

numofprevneurons = nopn;

Neurons = new Neuron[non];

double[,] Weights = WeightInitialize(MemoryMode.GET, type);

for (int i = 0; i < non; ++i) {

double[] temp\_weights = new double[nopn];

for (int j = 0; j < nopn; ++j)

temp\_weights[j] = Weights[i, j];

Neurons[i] = new Neuron(null, temp\_weights, nt); } }

protected int numofneurons;

protected int numofprevneurons;

Neuron[] \_neurons;

public Neuron[] Neurons { get => \_neurons; set => \_neurons = value; }

public double[] Data {

set {

for (int i = 0; i < Neurons.Length; ++i)

Neurons[i].Inputs = value; } }

public double[,] WeightInitialize(MemoryMode mm, string type) {

double[,] \_weights = new double[numofneurons, numofprevneurons];

Console.WriteLine($"{type} weights are being initialized...");

XmlDocument memory\_doc = new XmlDocument();

memory\_doc.Load($"{type}\_memory.xml");

XmlElement memory\_el = memory\_doc.DocumentElement;

switch (mm) {

case MemoryMode.GET:

for (int l = 0; l < \_weights.GetLength(0); ++l)

for (int k = 0; k < \_weights.GetLength(1); ++k)

\_weights[l, k] = double.Parse(memory\_el.ChildNodes.Item(k + \_weights.GetLength(1) \* l).InnerText.Replace(',', '.'), System.Globalization.CultureInfo.InvariantCulture);

break;

case MemoryMode.SET:

for (int l = 0; l < Neurons.Length; ++l)

for (int k = 0; k < numofprevneurons; ++k)

memory\_el.ChildNodes.Item(k + numofprevneurons \* l).InnerText = Neurons[l].Weights[k].ToString();

break; }

memory\_doc.Save($"{type}\_memory.xml");

Console.WriteLine($"{type} weights have been initialized...");

return \_weights; }

abstract public void Recognize(Network net, Layer nextLayer);

abstract public double[] BackwardPass(double[] stuff); }

# Додаток В

**Код файлу InputLayer.cs**

public class InputLayer

{

private readonly (double[], double[])[] \_trainset;

public InputLayer(DatabaseContext context)

{

var sets = context.TrainingSets.ToList();

var genreIds = context.GameGenres.Select(g => g.Id).ToList();

var tupleList = new List<(double[] aspects, double[] result)>();

foreach (var item in sets)

{

var aspects = context

.GameAspectGroups

.Where(a => a.GroupId == item.GameAspectGroupId)

.Select(a => a.Value)

.ToArray();

var result = new double[genreIds.Count];

result[genreIds.IndexOf(item.GameGenreId)] = 1;

tupleList.Add((aspects, result));

}

\_trainset = tupleList.ToArray();

}

public (double[], double[])[] Trainset { get => \_trainset; }

}

# Додаток Г

**Код файлу HiddenLayer.cs**

public class HiddenLayer : Layer {

public HiddenLayer(int non, int nopn, NeuronType nt, string type) : base(non, nopn, nt, type) { }

public override void Recognize(Network net, Layer nextLayer) {

double[] hidden\_out = new double[Neurons.Length];

for (int i = 0; i < Neurons.Length; ++i)

hidden\_out[i] = Neurons[i].Output;

nextLayer.Data = hidden\_out; }

public override double[] BackwardPass(double[] gr\_sums) {

double[] gr\_sum = null;

for (int i = 0; i < numofneurons; ++i)

for (int n = 0; n < numofprevneurons; ++n)

Neurons[i].Weights[n] += Constants.LearningSpeed \* Neurons[i].Inputs[n] \* Neurons[i].Gradientor(0, Neurons[i].Derivativator(Neurons[i].Output), gr\_sums[i]);

return gr\_sum; } }

# Додаток Д

**Код файлу OutputLayer.cs**

public class OutputLayer : Layer

{

public OutputLayer(int non, int nopn, NeuronType nt, string type) : base(non, nopn, nt, type) { }

public override void Recognize(Network net, Layer nextLayer)

{

for (int i = 0; i < Neurons.Length; ++i)

net.fact[i] = Neurons[i].Output;

}

public override double[] BackwardPass(double[] errors)

{

double[] gr\_sum = new double[numofprevneurons];

for (int j = 0; j < gr\_sum.Length; ++j)//вычисление градиентных сумм выходного слоя

{

double sum = 0;

for (int k = 0; k < Neurons.Length; ++k)

sum += Neurons[k].Weights[j] \* Neurons[k].Gradientor(errors[k], Neurons[k].Derivativator(Neurons[k].Output), 0);//через ошибку и производную

gr\_sum[j] = sum;

}

for (int i = 0; i < numofneurons; ++i)

for (int n = 0; n < numofprevneurons; ++n)

Neurons[i].Weights[n] += Constants.LearningSpeed \* Neurons[i].Inputs[n] \* Neurons[i].Gradientor(errors[i], Neurons[i].Derivativator(Neurons[i].Output), 0);//коррекция весов

return gr\_sum;

}

}

# Додаток Е

**Код файлу Neuron.cs**

public class Neuron

{

public Neuron(double[] inputs, double[] weights, NeuronType type)

{

\_type = type;

\_weights = weights;

\_inputs = inputs;

}

private NeuronType \_type;

private double[] \_weights;

private double[] \_inputs;

public double[] Weights { get => \_weights; set => \_weights = value; }

public double[] Inputs { get => \_inputs; set => \_inputs = value; }

public double Output { get => Activator(\_inputs, \_weights); }

private double Activator(double[] i, double[] w)

{

double sum = 0;

for (int l = 0; l < i.Length; ++l)

sum += i[l] \* w[l];

return Math.Pow(1 + Math.Exp(-sum), -1);

}

public double Derivativator(double outsignal) => outsignal \* (1 - outsignal);//f'(x)

public double Gradientor(double error, double dif, double g\_sum) => (\_type == NeuronType.Output) ? error \* dif : g\_sum \* dif;//g\_sum - это сумма градиентов следующего слоя

}