Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Метод нейромережевого налаштування автоматичного регулятора комп'ютерно-інтегрованної системи»

Виконав: студент 2 курсу, групи 3АКІТ-21м
спеціальності 151 – Автоматизація та
комп'ютерно-інтегровані технології

Фредерік ГУРАЛЬНИК
Ім'я ПРІЗВИЩЕ

Керівник:
к.т.н., доцент каф.АІІТ
степінь, звання, посада

Ілона БОГАЧ
Ім'я ПРІЗВИЩЕ
«10» 12 2022 р.

Опонент:
к.т.н., доцент каф. САІТ
степінь, звання, посада

Олексій КОЗАЧКО
Ім'я ПРІЗВИЩЕ
«12» 12 2022 р.

Допущено до захисту
Зав. кафедри КСУ
В'ячеслав КОВТУН
«14» 12 2022 р.

Вінниця ВНТУ – 2022 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра комп'ютерних систем управління
Рівень вищої освіти другий (магістерський)
Галузь знань – 15 – Автоматизація та приладобудування
Спеціальність – 151 – Автоматизація та комп'ютерно-інтегровані технології
Освітньо - професійна програма – Інформаційні системи і Інтернет речей

ЗАТВЕРДЖУЮ
Завідувач кафедри КСУ

В'ячеслав КОВТУН

"03" жовтня 2022 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ
студентці Гуральника Фредеріка Борисовича
(прізвище, ім'я, по батькові)

1. Тема роботи: Метод нейромережевого налаштування автоматичного регулятора комп'ютерно-інтегрованної системи
керівник роботи: к.т.н., доцент каф.АІІТ Богач Ілона Віталіївна
консультант роботи: д.т.н., доцент, зав. кафедри КСУ Ковтун В.В.
затверджені наказом ВНТУ від "14" вересня 2022 року № 203
2. Термін подання студентом роботи "12" грудня 2022 року
3. Вихідні дані до роботи: нейронна мережа, вибірка даних (відеозаписи), автокодер, мова програмування *Python/Matlab/Java/C#.*
4. Зміст текстової частини: вступ, аналіз предметної області, проектування нейромережевої реалізації автоматичного регулятора, розробка нейромережевого застосунку для розрахунку параметрів автоматичного регулятора, висновки, список використаних джерел.
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень) архітектура мережі; діаграма варіантів використання; робочий процес системи; приклад вихідних даних; лістинг програмного забезпечення.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-3	Ковтун В.В., д.т.н., доцент, зав. кафедри КСУ	03.10.22	08. 12.22

7. Дата видачі завдання "03" жовтня 2022 року

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Термін виконання		Примітк
		початок	закінчення	
1	Аналіз предметної області	03.10.22	20.10.22	
2	Проектування нейромережевої реалізації автоматичного регулятора	20.10.22	10.11.22	
3	Розробка нейромережевого застосунку для розрахунку параметрів автоматичного регулятора, висновки, список використаних джерел.	10.11.22	24.11.22	
4	Оформлення пояснювальної записки і графічного матеріалу	25.11.22	07.12.22	
5	Попередній захист роботи	12.12.22	16.12.22	
6	Остаточний захист роботи	20.12.22	23.12.22	

Студент _____ Фредерик ГУРАЛЬНИК
(підпис) (*Ім'я ПРІЗВИЩЕ*

Керівник роботи _____ Ілона БОГА
(підпис) (*Ім'я ПРІЗВИЩЕ*

АНОТАЦІЯ

УДК 004.81

Гуральник Ф. Б. Метод нейромережевого налаштування автоматичного регулятора комп’ ютерно-інтегрованної системи. Магістерська кваліфікаційна робота зі спеціальності 151 – Автоматизація та комп’ютерно-інтегровані технології, освітня програма – Інформаційні системи і Інтернет речей. Вінниця: ВНТУ, 2022. 115 с.

У магістерській кваліфікаційній роботі розроблено метод нейромережевого налаштування автоматичного регулятора комп’ ютерно-інтегрованної мережі. У оглядово-аналітичній частині роботи досліджено види і класифікації, математичні моделі регуляторів. Проаналізовано існуючі архітектури згорткових нейронних мереж. У теоретично-методичній частині розроблено рішення у вигляді нейронної мережі для налаштування регуляторів.Розроблений продукт аналізується та тестується.

Ключові слова: нейронна мережа, регулятори, алгоритм, додаток, комп’ ютерно-інтегровані системи.

ABSTRACT

UDC 004.81

F. B. Guralnyk. The method of neural network adjustment of the automatic controller of the computer-integrated system. Master's thesis on specialty 151 - Automation and computer-integrated technologies, educational program - Information systems and the Internet of Things. Vinnytsia: VNTU, 2022. 115 p.

In the master's qualification work, a method of neural network adjustment of the automatic controller of the computer-integrated network was developed. In the review and analytical part of the work, types and classifications, mathematical models of regulators are investigated. The existing architectures of convolutional neural networks are analyzed. In the theoretical-methodical part, a solution in theform of a neural network for adjusting the regulators is developed. The developed product is analyzed and tested.

Keywords: neural network, regulators, algorithm, application, computer integrated systems.

Відгук
керівника магістерської кваліфікаційної роботи

студента Гуральника Фредеріка Борисовича група ЗКІТ-21м
на тему: Метод нейромережевого налаштування автоматичного регулятора комп'ютерно-інтегрованої системи.

Актуальність роботи в контексті спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» доведена результатами інформаційного пошуку та аналізу літературних джерел. Додатковим підтвердженням актуальності роботи слугують опубліковані тези на Науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації.

Представлені в магістерській кваліфікаційній роботі рішення обґрунтовані послідовним ланцюгом дій, які включають пошук інформації про проблему, її узагальнення, формулювання прикладних задач для її вирішення, проектування відповідного програмного засобу для вирішення поставлених задач, його тестування і формулювання висновків. Раціональність та ефективність прийнятих рішень доведені результатами тестування.

Дипломник показав хороший рівень спеціальних знань і «м'яких» навичок. Дипломник продемонстрував вміння: - вирішувати поставлені керівником завдання самостійно, згідно власноруч розробленої схеми заходів; - здійснювати пошук і узагальнення інформації; - комунікативні навички. Доведенням ерудиції та креативності дипломника є вчасно представлена магістерська кваліфікаційна робота.

Основні результати, представлені в роботі отримані дипломником самостійно. Матеріалу роботи властивий високий ступінь оригінальності, що доведено результатами перевірники на наявність запозичень.

Дипломник працював ритмічно, без суттєвих відхилень від затвердженого графіку. Втрати зв'язку з керівником не було.

До недоліків можна віднести те, що дипломник навів лише необхідні і достатні діаграми, що описують процес проектування створеної системи. Результати багатоваріантного аналізу бажано було звести в таблицю. Автор не акцентує увагу на тім, які саме з отриманих результатів опубліковані у вигляді тез. Бажано було аналізувати свіжіші літературні джерела. Не всі рисунки достатньо якісні.

Загалом магістерська кваліфікаційна робота *відповідає* спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», заслуговує на оцінку А, а її автор *заслуговує* присудження кваліфікації: ступінь вищої освіти магістр, спеціальність «Автоматизація та комп'ютерно-інтегровані технології», освітня програма «Інтелектуальні системи і Інтернет речей».

Керівник магістерської кваліфікаційної роботи
к.т.н., доцент каф. АІТ Богач І. В.

Відгук
опонента на магістерську кваліфікаційну роботу

студента Гуральника Фредеріка Борисовича група ЗКІТ-21м
на тему: Метод нейромережевого налаштування автоматичного регулятора комп'ютерно-інтегрованої системи.

Актуальність роботи в контексті спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології» доведена результатами інформаційного пошуку та аналізу літературних джерел. Додатковим підтвердженням актуальності роботи слугують опубліковані тези на Науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації.

Перший розділ магістерської кваліфікаційної роботи повністю присвячений огляду літературних та інформаційних джерел за обраною темою. Проаналізовано не менш ніж три аналоги створюваної системи. Функціональність та форм-фактор створеної системи цілком визначені на основі результатів критичного огляду літератури.

Прийняті рішення обґрунтовані результатами огляду літератури, результатами проектування та втілені в функціонуючу програмну систему. Результати її тестування доводять правильність прийнятих рішень.

Експериментальні дослідження продумані і повні. Тести охоплюють як функції інтерфейсу створеної системи, так і доводять якість та повноту виконання нею функціонального призначення, обґрунтованого на етапі проектування.

Вміст графічної частина повною мірою репрезентує всі отримані в магістерській кваліфікаційній роботі результати. Якість рисунків в графічній частині прийнятна.

Недоліки: Варто було дослідити питання інтеграції створеної системи з відомою системою MathCad. Для тестування функцій системи можна було застосувати спеціалізовані автоматизовані середовища.

Загалом магістерська кваліфікаційна робота ***відповідає*** спеціальності 151 «Автоматизація та комп'ютерно-інтегровані технології», заслуговує на оцінку _____A_____, а її автор ***заслуговує*** присудження кваліфікації: ступінь вищої освіти магістр, спеціальність «Автоматизація та комп'ютерно-інтегровані технології», освітня програма «Інтелектуальні системи і Інтернет речей»

Опонент на магістерську кваліфікаційну роботу

Доцент каф. АІІТ _____ Козачко О. М.

ЗМІСТ

ВСТУП

В автонастроюванні регулятор – це пристрій, який виконує функцію підтримки заданої характеристики. Він виконує діяльність з управління або підтримки низки значень у системі.

Прикладами є регулятор напруги (це може бути трансформатор, якийможе регулювати відношення напруги трансформатора або електронна схема, яка виробляє певну напругу), регулятор тиску (наприклад, занурювальний регулятор), який підтримує вихідний сигнал при фіксованому тиску нижче ніж його вхід, регулятор палива (контроль подачі палива) тощо.

Універсального регулятора не існує, тому під кожну конкретну задачу необхідно підбирати регулятор і його параметри.

Для вибору параметрів використовуються різні методи, в тому числі експериментальні та математичні. Основним недоліком таких методів є їх відносна неточність і/або складність. Крім того, існує клас задач, які вимагають великих обчислювальних витрат і збільшують час вирішення, наприклад, в аеродинамічних задачах літальних апаратів на надзвукових швидкостях польоту, коли пов'язані з ними аеродинаміка, теплообмін і т.д.

Чисто теоретичні дослідження цих проблем ускладнюються складністю фізичних явищ і математичних інструментів, що використовуються для їх опису. Великі труднощі викликають і експериментальні дослідження в природних умовах. Фізичне моделювання широко використовується в лабораторних умовах, де важко створити адекватні моделі і вимагає використання дорогого обладнання. Вирішення таких завдань ускладнюється через вплив нерозрахованих параметрів, які можуть спотворити кінцевий результат. Тому деякі проблеми вирішуються на основі інтуїції та логічного мислення. Коли через високу складність об'єкта управління неможливо врахувати вплив багатьох факторів, цей метод прогнозування дає низьку точність, а іноді навіть прямо протилежний результат.

Для вирішення задачі визначення параметрів авторегулятора можуть бути використані спеціалізовані системи, що реалізують нейромережеву техніку. Важливою відмінністю цих систем є те, що характер обробки інформації залежить не від визначеного алгоритму, а від розподілу зв’язків між нейронами, тому така система може адаптуватися до нових даних і швидко адаптуватися до нових вхідних даних. Крім того, стан кожного окремого нейрона визначається впливом стану багатьох інших підключених до нього нейронів, тому втрата одного чи кількох зв’язків не вплине суттєво на роботу всієї системи, забезпечуючи таким чином високу надійність система пол.

На практиці при розробці таких нейронних мереж проектується певна модель об'єкта, визначаються вхідні та вихідні параметри, і після навчання нейронної мережі можна отримати результати з досить високим ступенем точності. Ця точність залежить від правильного вибору вхідних і вихідних параметрів, вибору топології мережі, кількості шарів нейронів тощо. Однак, при правильному виборі вищевказаних параметрів нейронної мережі, нейронна мережа може передбачити параметри не тільки однієї заздалегідь заданої задачі, а й багатьох схожих завдань.

Метою даної магістерської роботи є покращення методів застосування нейронних мереж для визначення параметрів генератора, а саме створення програми, яка може бути використана для створення нейромережевих додатків для визначення параметрів авторегулятора, що на відміну від існуючих дозволить розрахувати параметри генератора автоматично без кропіткого налаштування моделі.

Для виконання завдання необхідно:

- дослідити проблему визначення параметрів авторегулятора аналітичними, математичними та експериментальними методами;

– дослідити існуючі рішення для створення нейронних мереж;

- вирішити конкретні прикладні завдання, які необхідно вирішити за допомогою нейронних мереж;

- визначити топологію та основні параметри нейронної мережі;

– виконати процедуру створення нейронної мережі за визначеною топологією та параметрами;

– за допомогою створеної програми згенерувати нейронну мережу, яка визначатиме параметри авторегулятора;

- Навчати ШНМ та аналізувати результати роботи нейронних мереж.

Об'єкти та теми дослідження. Об’єктом дослідження даної роботи є нейронні мережі. Предметом розробки та дослідження є застосування нейронних мереж для визначення параметрів авторегулятора.

Методи дослідження. Досліджує методи визначення параметрів авторегулятора за допомогою математичних і аналітичних методів; пропонує топологію нейронної мережі та параметри для вирішення завдання; програмно реалізує багаторівневий персептрон для навчання ШНМ з використанням методів помилкового зворотного поширення. Використовується для проектування, написання, налагодження коду та тестування програмних продуктів .

Наукова новизна роботи полягає в використанні топології «багатошарового персептрона» і інверсного методу розповсюдження помилок для визначення параметрів автогенератора, що на відміну від існуючих дозволяє розрахувати параметри генератора автоматично без кропіткого налаштування моделі.

Практичне значення отриманих результатів. Розроблено програму для генерації та навчання нейронних мереж з топологією «багатошарового персептрона» та інверсним методом розповсюдження помилок. Ця програма дозволяє створювати ШНМ з будь-якою кількістю шарів на нейрон. Функції активації можна вибрати окремо для кожного з чотирьох доступних рівнів, що дозволяє генерувати різні комбінації нейронних мереж. Ця програма використовується для створення нейронної мережі для визначення параметрів автоматичного регулятора водопостачання та нейронної мережі для визначення кількості регуляторів, які необхідно використовувати в динамічній системі керування за заданим відхиленням.

Апробація результатів роботи. Результати роботи було представлено на L науково-технічній конференції факультету комп’ютерних систем і автоматики Вінницького національного технічного університету, Вінниця, 2021 [25, 26]; прийнято тези на Всеукраїнську науково-практичну інтернет-конференцію студентів, аспірантів та молодих науковців «Молодь в науці: Дослідження, проблеми, перспективи (МН-2023) та продано документи на отримання свідоцтва на авторське право.

1 АВТОМАТИЧНІ РЕГУЛЯТОРИ ЯК ОБ’ЄКТ ДОСЛІДЖЕННЯ

1.1 Класифікація автоматичних регуляторів

Система автоматичного керування (САУР) – це така система автоматичного керування, завданням якої є утримання початкового значення Y(t) об’єкта на заданому рівні або зміна його за заданим законом.

Система автоматичного регулювання складається з об’єкта регулювання та елементів керування, які впливають на об’єкт при зміні однієї чи кількох змінних, що маніпулюють. Керована змінна змінюється під впливом вхідного сигналу (керування чи збурення). Метою регулювання є формування законів, у яких вихідні регуляторні змінні не сильно відрізняються від необхідних значень. У багатьох випадках вирішення цього завдання ускладнюється наявністю випадкових збурень (перешкод). При цьому необхідно вибрати такий закон керування, щоб при проходженні керуючого сигналу через систему спотворення були невеликими, а шумовий сигнал майже не проходив.

Автоматичне налаштування забезпечує оптимальну роботу динамічних систем, значно підвищуючи продуктивність праці та усуваючи необхідність виконувати одні й ті самі завдання знову і знову.

Авторегулятор – пристрій, складова частина системи авторегулювання, що формує керуючі сигнали для зміни (регулювання) вихідних параметрів.

Автоматизована система нагляду включає також наглядовий об’єкт – систему, яка потребує наглядової дії для роботи в заданому режимі.

Коригованим об'єктом може бути будь-яка динамічна система або її модель. Стан об'єкта характеризується деякими кількісними значеннями, які змінюються з часом, а саме змінними стану. У природному процесі ці змінні можуть діяти як температура, щільність речовини в тілі, обмінний курс цінного паперу тощо. Для технічних об'єктів це механічні рухи (кутові чи лінійні) та їх швидкість, електричні величини, температура тощо. Аналіз і синтез систем

керування здійснюється за допомогою засобів теорії управління, спеціального розділу математики.

Авторегулятор виконує завдання, визначені системою керування. Наглядовий орган приймає рішення щодо наглядової дії та діє на об’єкті регулювання через виконавчі органи та наглядовий орган. Під час роботи системи керування чутливий механізм безперервно вимірює величину регулювання, а регулятор може впливати на виконавчий механізм безперервно або мати переривчасту (дискретну) дію. Вони поділяються на прямі (рисунок 1.1) і непрямі (рисунок 1.2) регулятори [2] залежно від характеру регулюючої дії на об’єкт регулювання або конструкції регулятора.

Рисунок 1.1 - Структурна схема типового автоматичного регулювання

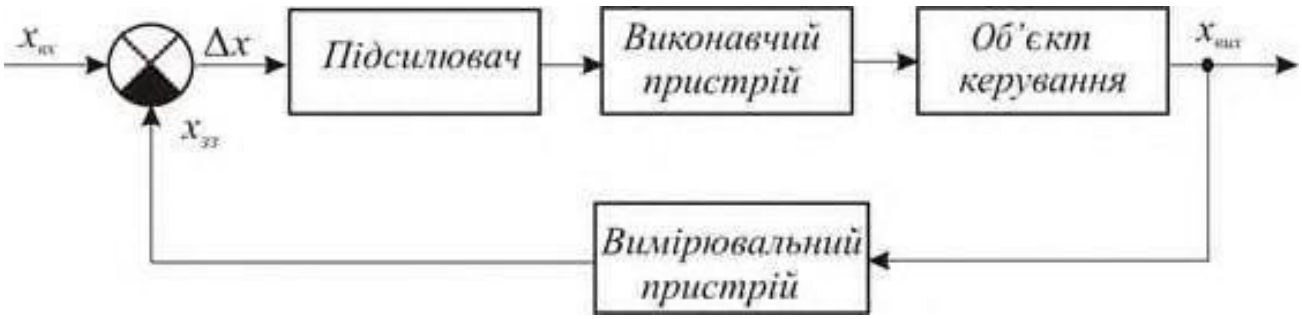


Рисунок 1.2 - Структурна схема непрямого автоматичного регулювання

До регуляторів прямої дії відносяться регулятори, в яких зусилля, необхідне для переміщення регулятора, виникає без надання додаткової енергії за рахунок зміни вихідного параметра. У цьому випадку датчик (чутливий елемент) є одночасно і виконавчим механізмом.

При прямому регулюванні автоматичний регулятор має один ступінь свободи. Він складається з елемента, який реагує на зміну регульованої величини і діє безпосередньо на регулятор об'єкта регулювання.

На практиці більш широке застосування отримали непрямі регулятори. Серед цих регуляторів необхідно мобілізувати зусилля регуляторів через вплив окремих механізмів примусу. Ці регулятори класифікуються за типом енергії, яка використовується для переміщення приводу:

- Механічні;
- Пневматичні;
- гідравлічний;
- електричні;
– комбайнувати;
– Змішування, наприклад, пневматична гідравліка.

Залежно від кількості настроюваних режимів можна виділити:

- однорежимний;
– два режими;
– повний режим;
- загальні.

Крім того, стабілізатор напруги підрозділяється на релейний, безперервний і імпульсний.

Релейні регулятори також відомі як регулятори положення.

Регулятори також класифікуються як екстремальні та стабільні. Екстремальні модифікатори можна використовувати для об’єктів із надзвичайно статичними властивостями, зовнішній вигляд яких може залежати від зовнішнього впливу та змінюватися з часом.

Найбільш поширеним є регулятор-стабілізатор, тобто регулятор, який регулює роботу регульованого об'єкта так, щоб відхилення не перевищувало заданого значення.

Для контролю безперервної модуляції використовуються контролери зі зворотним зв'язком для автоматичного керування процесом або операцією.

Система керування порівнює значення або стан контрольованої змінної процесу з бажаним значенням або заданим значенням і використовує різницю як керуючий сигнал, щоб зробити вихідну змінну процесу установки таким самим значенням, як задане значення.

Часова залежність керуючого сигналу, що формується регулятором, від незбалансованого сигналу визначається законом регулювання, з яких найбільш широко використовуються:

– пропорційний метод регулювання;

– загальне нормативне право;

– пропорційно-інтегральний метод регулювання;

– пропорційно-різницевий метод регулювання;

– Пропорційно-інтегрально-похідний метод регулювання.

Відповідно, за законом регулювання регулятори напруги поділяються на інтегральні (І), пропорційні (П), пропорційно-інтегральні (ПІ), пропорційно-похідні (ПД) і пропорційно-інтегрально-похідні (ПІД).

Регулятивні закони формуються за допомогою зворотного зв'язку. Враховуючи динамічні характеристики об'єкта керування, він визначає тип і якість перехідного процесу САР.

Контролери зворотного зв'язку іноді називають контролерами жорсткого зворотного зв'язку. Крім того, існують регулятори з комбінованим зворотним зв'язком. У літературі комбінований зворотний зв’язок ще називають ізотропним зворотним зв’язком.

Регульований об'єкт, незалежно від його типу (тепловий двигун, теплова машина, парова машина), обладнаний автоматичним регулятором (системою автоматичного регулювання), має свої статичні та динамічні характеристики. Ці особливості можна описати відповідними залежностями.

За типом статичних (нормативних характеристик) SAR поділяють на статичні та нестатичні.

Статичні системи регулювання включають системи, які підтримують значення параметра регулювання в деяких чітко визначених межах Δh або Δn, коли регулятор δ не є рівномірним.

Нестатичні системи керування підтримують лише постійне значення n-го параметра налаштування. Наприклад, безстатична природа дизельної електростанції забезпечує постійну частоту струму 50 Гц при постійній частоті обертання колінчастого вала двигуна (n=const). SAR, що характеризується одним регулятором і одним регульованим значенням, називається одноімпульсною системою.

Подвійним імпульсом називається авторегулятор, який реагує не тільки на відхилення параметра регулювання n (кількості обертів), але і на його похідну ω - прискорення: n, ω.

У системах, де регулюється кілька параметрів одночасно, наприклад в паровій турбіні, контролюються і регулюються кілька параметрів: обертання ротора турбіни, тиск пари, температура пари на вході і виході та інші електростатичні параметри.

1.2 Математична і графічна моделі процесу автоматичного регулювання

Математично процес регулювання характеризується зміною величини регулювання x1 в момент часу t. Процес регулювання x1(t) можна представити у вигляді графіка, який також називають кривою процесу регулювання. Приклад такої діаграми показано на рисунку 1.3.

За відсутності зовнішнього та внутрішнього збудження АСУ в ідеальному випадку отримаємо пряму x 0(t), як показано на рис.1. 1.3 позначеноштриховою лінією. У реальних умовах роботи подразник постійно діє на систему. При цьому крива процесу коригування x1(t), що відображає фактичне значення величини коригування, повинна бути близькою до прямої x1(t).

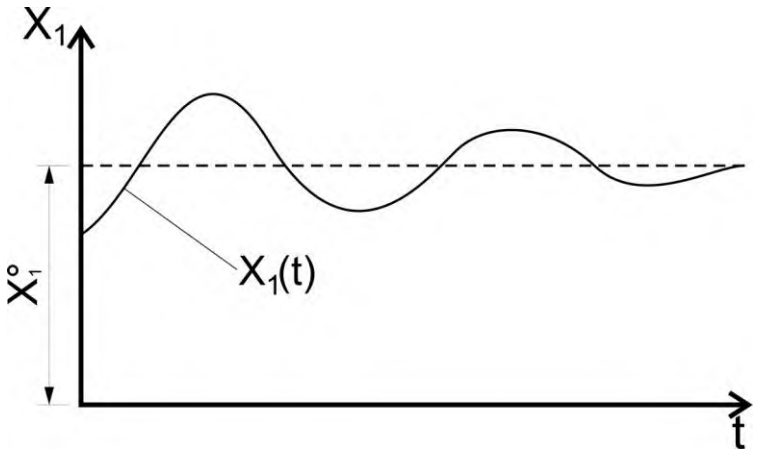


Рисунок 1.3 - Графік регулювання

Це означає, що незалежно від того, яке збудження діє на систему, авторегулятор повинен завжди підтримувати параметр налаштування (кількість) навколо заданого значення x 0. Крива процесу кондиціонування показує здатність даної системи справлятися зі своїм завданням. Технічна вимога SAR – це межа, яку не може перевищувати фактичне значення зазначеної величини, тобто відхилення кривої x1(t) 0 0 від заданого значення прямої x1(t). Наприклад, на 2% від значення x1.

1.3 Параметризація процесу автоматичного регулятора

Параметри регулятора та його ланок включають:

– неоднорідність регулятора δ;

- коефіцієнт підсилення;

– передавальне число;

– час затримки передачі сигналу;

- Момент інерції;

– індуктивність;

- місткість;

– Резистори електричного з’єднання та інші резистори.

Відповідність вищевказаних параметрів регулятора заданим параметрам ще не гарантує, що регулятор придатний для виконання своїх функцій. У разі

невдалого вибору параметра регулятор може змусити SAR не заспокоїти систему, а навпаки - потрясти її. При цьому крива процесу регулювання буде відхилятися від заданого значення (рис. 1.4, 1.5).

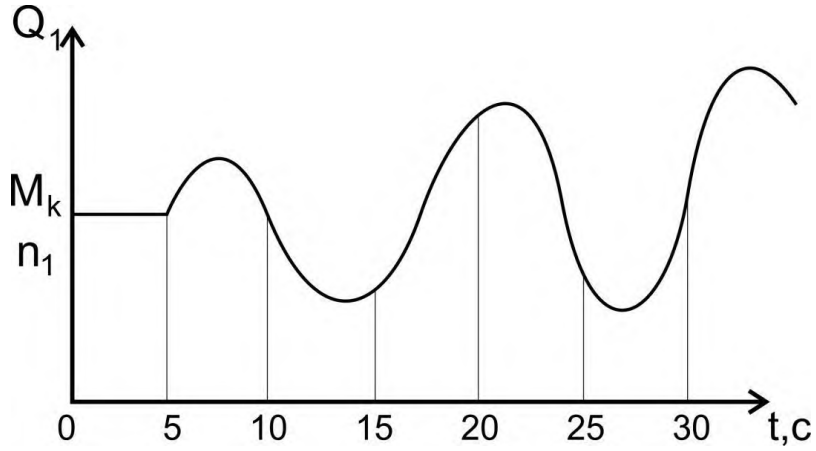


Рисунок 1.4 - Нестійка регульована система (траєкторія її руху розходиться).

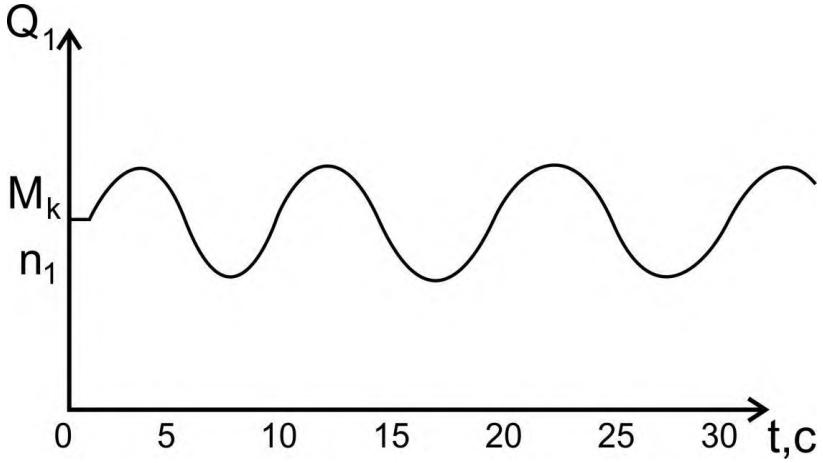


Рисунок 1.5 - Нестійка регульована система (з постійною амплітудою коливань).

Потрібно не тільки пройти експериментальний розрахунок і перевірку Рівноважні режими роботи машинних ЗАР, а також перехідні процеси та інші динамічні режими, коли на об'єкт кондиціонування діють змінні збурення, через які він заважає ЗАР. Під час різних типів перехідних процесів необхідно знати наступне:

– при включенні машини та SAR;

– при переналаштуванні системи на нове значення регульованої величини;

– збудлива дія у вигляді стрибкоподібних змін навантаження системи;

- при збуджуючих рухах або заданих навантаженнях;

- Коли SAR увімкнено, через роботу регулятора поточне значення маніпулюваного значення (параметра) може мати велике початкове відхилення від значення, яке необхідно встановити.

Щоб швидко усунути великі початкові відмінності між двома параметрами, необхідно швидко відрегулювати регулятор на початку процесу. Це відхилення маніпуляційного значення в протилежний бік називається люфтом (перенастроюванням). Технічні вимоги SAR визначають швидке згасання перехідних процесів, які можуть вплинути на якість регульованої системи.

1.4 Визначення нейрона та нейромережі

Нейронна мережа (штучна нейронна мережа) – це метод, який використовує фізичне апаратне забезпечення або комп’ютерне програмне забезпечення для моделювання обчислювальних властивостей, подібних до тих, які припускають реальні нейронні мережі, наприклад здатність навчатися та підтримувати зв’язки. Нейронні мережі можуть плавно апроксимувати та інтерполювати дані з багатьох змінних у компактний спосіб, який інакше може вимагати величезних баз даних.

Нейронні мережі мають велику кількість процесорів (рівнів). Ці процесори працюють паралельно, але пошарово. Перший рівень отримує необроблені дані, подібно до того, як зоровий нерв людини отримує необроблену інформацію. Потім кожен наступний шар отримує вхідні дані від попереднього шару та передає вихідні дані шару після нього. Останній шар визначає кінцевий результат.

Маленькі вузли складають кожен рівень. Вузли тісно пов’язані з вузлами на попередньому та наступному рівнях. Кожен вузол нейронної мережі має власну область знань, включаючи правила, які він пише, і правила, які він вивчає самостійно. Ключовим фактором ефективності нейронних мереж є те,що вони надзвичайно адаптивні та дуже швидко навчаються. Кожен вузол оцінює важливість вхідних даних, які він отримує від попередніх вузлів. Вхідні дані, які найбільше сприяють правильному виходу, отримують найбільшу вагу.

Нещодавня популярність нейронних мереж у різних сферах пояснюється, зокрема:

– Дані – Величезна кількість доступних даних, зібраних за останнє десятиліття, значною мірою сприяла популярності глибокого навчання. Це дозволяє ШНМ справді продемонструвати свій потенціал, оскільки вони визначають результати з меншою кількістю помилок, коли кількість даних збільшується;

- Обчислювальна потужність - обчислювальна потужність, доступна в останні десятиліття, дозволила обробляти більше даних з більш високою швидкістю, що призвело до вдосконалення та прискорення навчання ШНМ;

- Алгоритми - Успіхи в області розробки алгоритмів дозволяють вибрати відповідну топологію нейронної мережі для конкретного завдання, а розроблені методи навчання дозволяють швидше навчати нейронні мережі;

- Маркетинг. Незважаючи на те, що нейронні мережі існують з 1944 року, вони набули справжньої популярності після створення кількох успішних продуктів, таких як робота Софії в Hanson Robotics, створення якої призвело до збільшення охоплення цієї теми в ЗМІ.

ШНМ складається з нейронів і зв’язків між ними. У різних топологіях нейронної мережі використовується різна кількість нейронів і шарів нейронів, а також різні зв’язки між ними (наприклад, у повністю рекурентній ШНМ кожен нейрон у мережі з’єднаний з кожним іншим нейроном у мережі) нейрони, а в перцептроні , кожен нейрон шару з’єднаний з кожним нейроном попереднього шару).

Нейрон – це компонент ШНМ, який обчислює задану функцію суматора, виконуючи операції над вхідними даними, і Повертає результат як вихід нейрона. Нейрони одного рівня згруповані в шар.

Зв'язки між нейронами мають певну вагу - результат обчислення нейронів попереднього шару потрібно помножити на це число. Потім результат множення передається на вхід наступного шару нейронів. Тим часом, у більшості випадків ваги з’єднання є єдиними параметрами, які змінюються під час навчання ШНМ. Інші параметри мережі, такі як функція суматора та функція активації нейронів, залишаються незмінними.

1.5 Підходи до визначення параметрів регулятора за допомогою ШНМ

Поряд із системами нечіткої логіки нейронні мережі на сьогодні є найперспективнішим напрямком розвитку систем керування складними електротехнічними системами. Ці галузі розвитку систем управління відносяться до категорії інтелектуальних систем управління, тому хороші результати можна отримати, коли математична модель об'єкта управління є неповною або параметри об'єкта управління змінюються під час роботи. .

Останнім часом з'являється все більше робіт з використання нейронних мереж для реалізації систем автоматичного керування. Однак більшість систем, які вони розглядають, так чи інакше повторюють класичний підхід впровадження регуляторів на основі класичних законів регулювання, де керуючий ефект пропорційний відхиленню, інтеграл або похідна відхилення величини регулювання. його встановлене значення. Водночас дуже перспективним вважається інший спосіб побудови регуляторів – шляхом передбачення значення параметра регулювання, яке буде отримано через певний час після можливого заданого впливу на систему регулювання.

Рішення про вплив приймаються на основі порівняння прогнозованих і очікуваних значень контрольних параметрів. Вирішення задач прогнозування

спирається на нейронні мережі, попередньо налаштовані на апроксимацію параметрів системи, з можливістю адаптивного уточнення.

В даний час існує багато мереж, призначених для управління конкретними системами. Однак дані ANN здебільшого мають закритий доступ ірозроблені для конкретних систем. Для кожного завдання необхідно розробити власну нейронну мережу, оскільки параметри системи та регулятора можуть відрізнятися від системи до системи.

При розробці штучних нейронних мереж використовуються різні програмні системи, які в основному включають багато варіантів топологій і параметрів штучних нейронних мереж, однак штучні нейронні мережі, створені за допомогою таких програмних систем, не завжди можна використовувати поза програмною системою як окремий модуль використання.

1.6 Аналіз існуючих рішень

В Інтернеті можна знайти достатньо теорій про створення нейронних мереж для регулювання процесів чи систем, але складно знайти готові рішення. Більшість нейронних мереж, призначених для управління або визначення параметрів авторегулятора, розроблені для конкретних завдань і знаходяться в закритому стані доступу.

Для створення штучних нейронних мереж використовуються різні програмні системи, але всі вони мають певні обмеження - неможливе створення окремих модулів нейронних мереж, вони містять надлишкові типи нейронних мереж, не дають можливості вводити власні функції підсумовування, вони мають обмежений набір функцій активації тощо. Розглянемо кілька таких систем.

Neuroph – це система розпізнавання нейронної мережі, написана мовою Java (рис. 1.6). Він складається з Java API, який включає базові класи, утиліти та реалізації певних типів нейронних мереж. Він також включає середовище Neuroph Studio, реалізоване на платформі NetBeans.

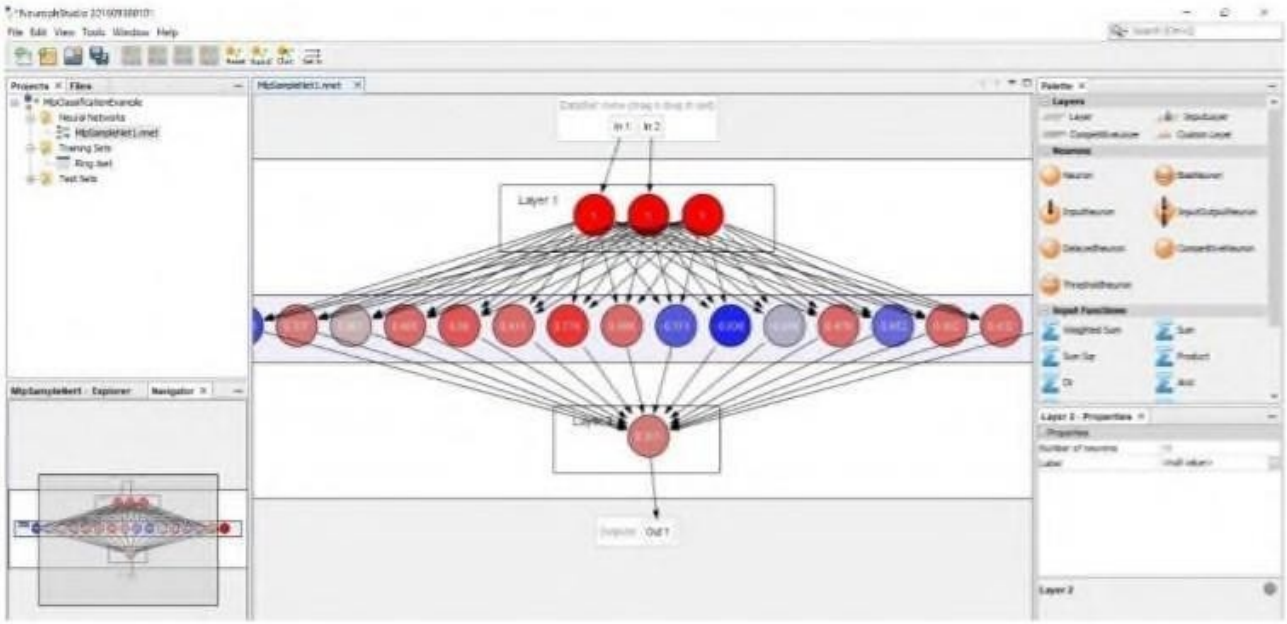


Рисунок 1.6 - Програма Neuroph

У середовищі реалізовані наступні архітектури нейронної мережі:

– Адалін;

- датчик;

– багатошаровий персептрон з алгоритмом зворотного поширення помилки, момент;

– мережа Хопфілда;

– двостороння асоціативна пам'ять;

– Мережа Кохонена;

– Мережа Hebb;

– Нейронна мережа RBF.

Система містить приклади прогнозування цін на фінансовому ринку за допомогою Інтернету, приклади класифікації тварин тощо. Пакет досить складний і містить багато зайвої інформації. Крім того, він громіздкий, оскільки містить модулі для створення різних типів нейронних мереж і не може використовуватися як окремий модуль.

Simbrain – це кросплатформна система, написана мовою Java (рис. 1.7). Система зосереджена на видимості та простоті. Містить реалізацію 2D-світу з різними об’єктами для тестування нейронних мереж для контролю.

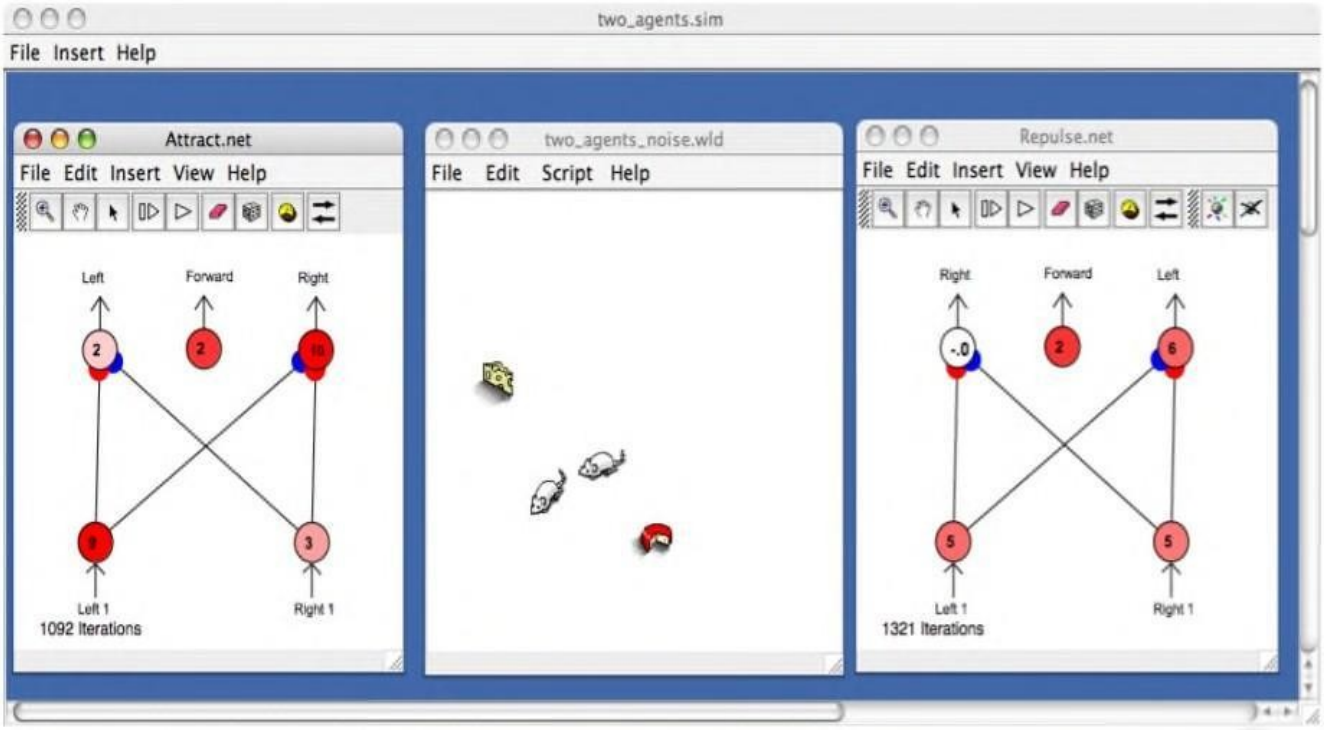


Рисунок 1.7 - Програма Simbrain

Система реалізує велику кількість різних варіантів нейронів і типів мережі може навчатися без вчителя. Створення та редагування нейронних мереж у середовищі Simbrain є візуальним. У разі візуальної побудови користувачі можуть бачити процес зміни ваги та передачі сигналу в мережі. Алгоритми зворотного поширення також можна відстежувати. Але користувачі також можуть використовувати Java API для створення власних завдань і додавання необхідної технології до системи. Зразки вводяться в мережу вручну, що не завжди зручно. Система більше призначена для навчання, ніж для створення робочих модулів. Складність введення навчальних даних і великий розмір програми через наявність бібліотек візуалізації робить програму непридатною для вирішення поставленої задачі.

MATLAB Deep Learning Toolbox (раніше Neural Network Toolbox) – це розширення MATLAB, яке містить інструменти для проектування, моделювання, розробки та візуалізації нейронних мереж (рис. 1.8). Пакет

забезпечує повну підтримку типових парадигм нейронних мереж і має відкриту модульну архітектуру. Цей пакет містить функції командного рядка таграфічний інтерфейс користувача для швидкого покрокового створення нейронних мереж. Крім того, Deep Learning Toolbox забезпечує підтримку Simulink, що дозволяє моделювати нейронні мережі та створювати блоки на основі розроблених структур нейронних мереж.

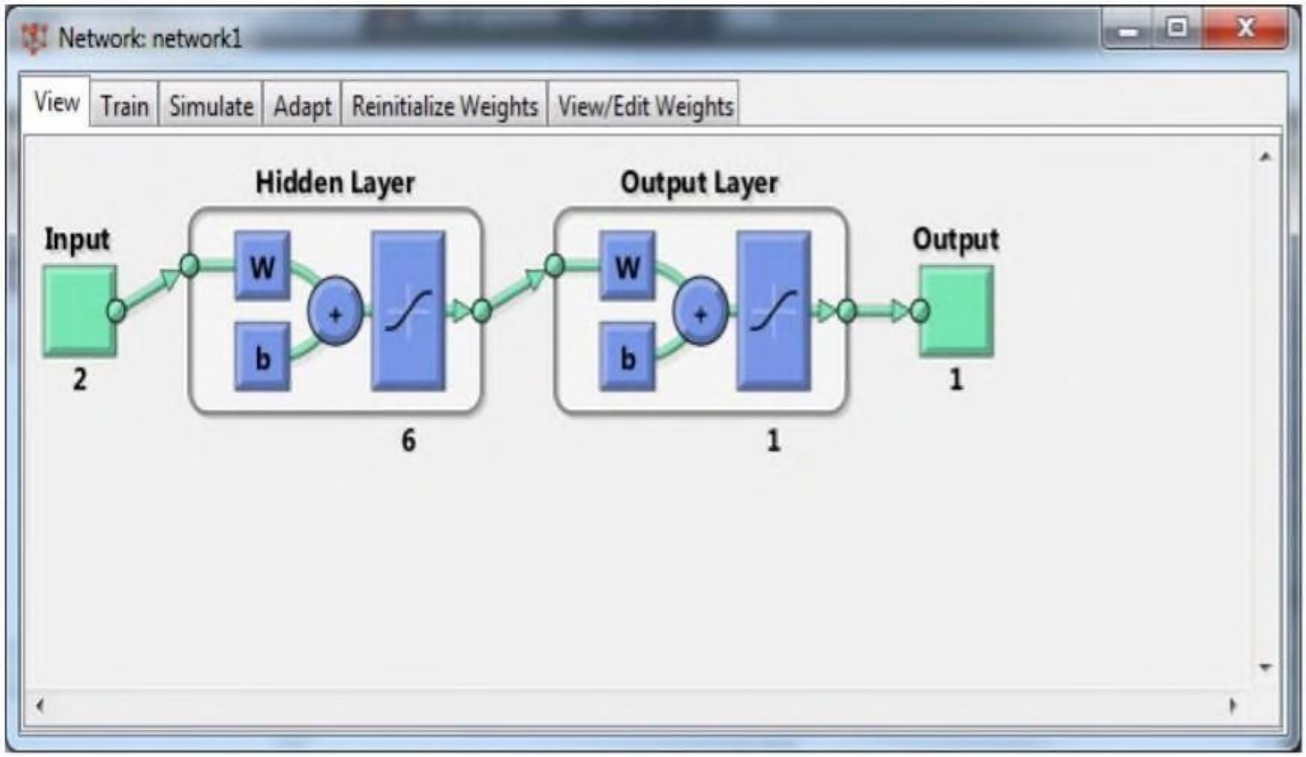


Рисунок 1.8 - Модуль Deep Learning Toolbox в середовищі MATLAB

Основні особливості цього модуля:

– графічний інтерфейс користувача для покрокового створення, навчання та імітації моделювання нейронної мережі;

– підтримує найбільш поширені керовані та некеровані мережеві структури;

– повний перелік функцій навчання та тестування;

– Алгоритми динамічного навчання мережі, включаючи затримку часу, нелінійну авторегресію (NARX), ланцюгові та регульовані динамічні структури;

– Блоки Simulink для створення нейронних мереж і блоки високого рівня для керування системами;

– Автоматична генерація блоків Simulink з об’єктів нейронної мережі;

– модульне представлення мережі, що дозволяє створювати необмежену кількість вхідних рівнів і комбінованих мереж, а також графічне представлення архітектури мережі;

– Функції попередньої та постобробки та блоки Simulink для покращення процесу навчання та оцінки продуктивності мережі;

- Візуалізація топології нейронної мережі та процесу навчання.

Як і попередня система, ця система має надлишкові модулі, що збільшує загальний розмір програми. Крім того, хоча цей фреймворк підтримує більш широкий спектр нейронних мереж, це комерційний проект, тому він не безкоштовний.

Encog. Система Encog є найповнішою з розглянутих систем (рис. 1.9). Він містить велику кількість ефективно написаних класів Java. На відміну від інших систем, вона реалізує не тільки алгоритми розпізнавання нейронних мереж, але й кластеризацію, генетичні алгоритми, приховані марковські моделі, байєсовські мережі тощо. Підтримувані архітектури нейронних мереж включають Adaline, машини Больцмана, нейронні мережі зворотного поширення, рекурентні мережі Елмана, рекурентні мережі Джордана,самоорганізуючі графіки Кохонена тощо. Доступні такі функції активації нейронів: біполярна, конкурентна, Елліотта, Гаусса, гіперболічна дотична, лінійна, синусова, сигмоїдна. Порівняно з іншими системами, система Encog написана як дуже ефективна, що дозволяє мережі працювати паралельно на машинах із кількома процесорами.

Усі перераховані реалізації мають багато доступних топологій нейронної мережі, але не дозволяють додавати нові мережеві архітектури, змінювати властивості навчання або оптимізувати мережеві структури. Системи з широкою конфігурацією, які обробляють різноманітні завдання та містять

велику кількість доступних архітектур нейронних мереж, не можуть бути безпосередньо застосовані до певного завдання.

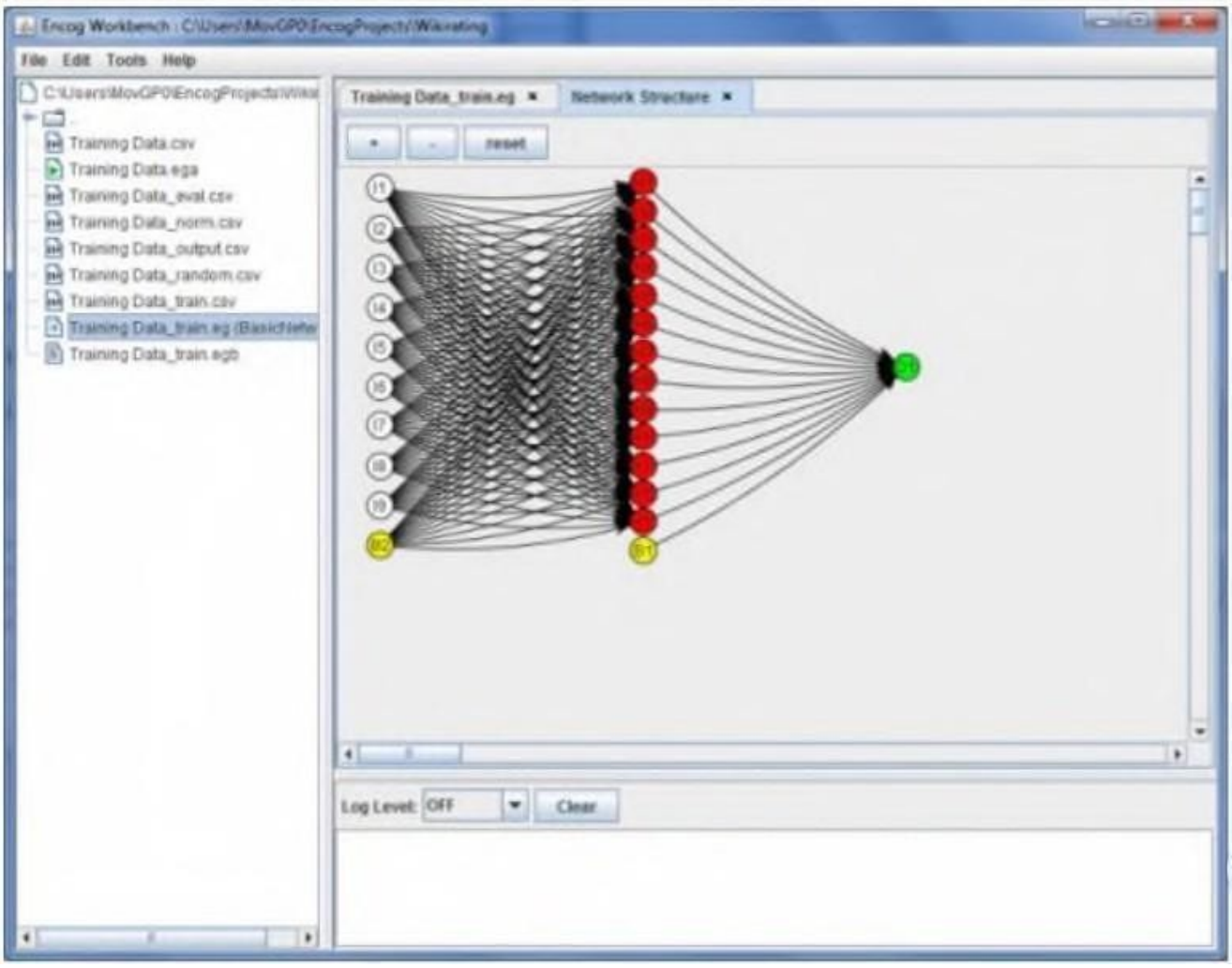


Рисунок 1.9 - Програма Encog

У розділі дипломної роботи розглядаються основи автомасштабувальників. Автоматичний регулятор необхідний для стабілізації роботи системи від зовнішніх і внутрішніх впливів. Оскільки сила і напрям зовнішніх впливів можуть змінюватися, регулятор повинен мати можливість стабілізувати систему під час кожного її запуску. Для нормальної роботи САР необхідно підібрати оптимальні параметри регулятора, тому що вибір параметрів залежить від того, стабілізує систему авторегулятор або дає більші похибки. Для вибору параметрів можна використовувати штучну нейронну мережу. В даний час існують нейронні мережі, які вибирають параметри

авторегулятора, але такі системи можуть використовуватися тільки для однієї задачі і здебільшого є закритими.

Розглянуто існуючі програмні системи для створення нейронних мереж, але більшість із них мають великі обсяги та велику кількість надлишкових функцій, а нейронні мережі, створені в таких програмах, не можуть бути використані в окремому прикладному використанні.

2 ПРОЕКТУВАННЯ НЕЙРОМЕРЕЖЕВОЇ РЕАЛІЗАЦІЇ АВТОМАТИЧНОГО РЕГУЛЯТОРА

2.1 Постановка задачі

Для прикладу витягнемо перше питання з п’ятого підрозділу другої частини методичного посібника [6]. Для забезпечення жителів села водою була побудована водонапірна башта, в яку насосом закачувалась річкова вода. Кожен споживач може в будь-який час включити воду на своїй ділянці, наприклад, для поливу. Необхідно побудувати систему автоматичної підтримкизаданого рівня води h0 (в метрах) в резервуарі.

Для простоти припустимо, що ця водонапірна вежа є резервуаром з водою. У дні ємності просвердлюється отвір, через яке тече вода. Через Sпозначимо площу поперечного перерізу бака. Нехай швидкість течії рідини q змінюється незалежно (в теорії управління це називається навантаженням на тіло).

Ми припускаємо, що споживачів досить багато, тому завжди хтось включає воду і насос постійно працює, щоб закачати її в бак. Щоб контролювати рівень води h, ми можемо змінювати її витрату Q (у м3/с). Таким чином, рівень h є величиною регулювання, а швидкість потоку Q є керуючим сигналом. Для зворотного зв’язку ми використовуємо датчик для вимірювання рівня води h в резервуарі (для простоти ми не будемо вдаватися в деталі типу датчика та швидкості, ми просто припустимо, що датчик вимірює кожну секунду і що налаштування значення насоса також відбувається кожну секунду).

Побудуємо математичну модель об’єкта, а саме танка. Вихідна витрата q (в м3/с) показує, скільки води витікає з резервуара за 1 секунду - це навантаження (рисунок 2.1).

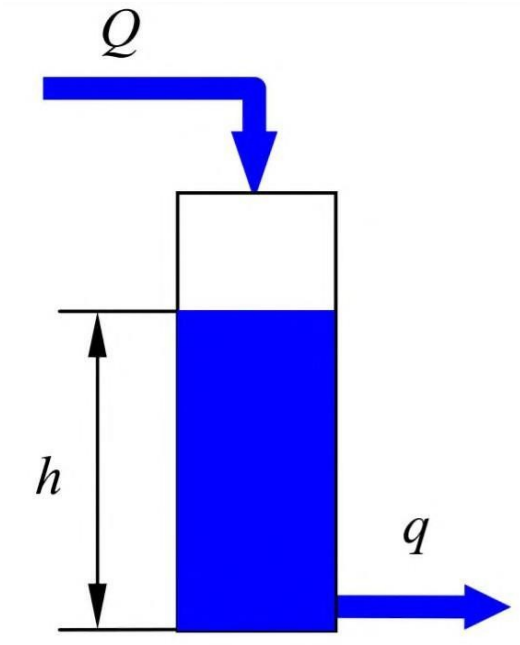


Рисунок 2.1 - Графічна інтерпретація цистерни

Зміна рівня Δh залежить від різниці витрат Q - q і площі поперечного перерізу резервуара S.

Якщо різниця витрат постійна протягом інтервалу часу Δt, то

$$\Delta Q(t) = \frac{Q(t) - q(t)}{S} * \Delta t. \quad (2.1)$$

Загалом, потрібно використовувати бали:

$$\Delta Q(t) = \frac{1}{S}\int_0^t (Q(t) - q(t))\, dt. \quad (2.2)$$

Нехай рівень води в момент часу t = 0 дорівнює заданому значенню, вв

Вихідний потік дорівнює ($Q(0) = q(0) = q0$), тому рівень не змінюється. Цей режим візьмемо за номінальний (робочу точку). Щоб отримати рівняння відхилення, уявімо потік у формі

$$Q(t) = q_0 + \Delta Q(t), \tag{2.3}$$

де ΔQ(t) – мале відхилення вхідної витрати від номінального режиму.

$$q(t) = q_0 + \Delta q(t), \tag{2.4}$$

де Δq(t) – мале відхилення вихідної витрати від номінального режиму.

Тоді, опустивши знак збільшення Δ, можна записати модель об’єкта керування у вигляді:

$$h(t) = \frac{1}{S}\int_0^t (Q(t) - q(t))\, dt, \tag{2.5}$$

де h(t) – відхилення між рівнем води в резервуарі та номінальним значенням;

Q(t)—відхилення вхідної витрати від номінального;

q(t) – відхилення вихідної витрати від номінального значення.

Для простоти приймемо S = 1 м2. Тобто при навантаженні q=0,5 м3/с об’єм води в резервуарі за одну секунду зменшиться на 0,5 м3, тому висота стовпа води зменшиться на 0,5м. В якості зворотного зв'язку ми будемо використовувати сигнал датчика рівня рідини. Похибка контролюрозраховується як різниця між заданим рівнем води та виміряним рівнем води:

$$e(t) = h_0(t) - h(t). \tag{2.6}$$

Застосуємо найпростіший з можливих регулятор - підсилювач (або пропорційний регулятор, Р-регулятор) з коефіцієнтом К, який регулює витрату за формулою:

$$Q(t) = K * e(t) = K * [h_0(t) - h(t)]. \quad (2.7)$$

Блок-схема системи керування представлена на рисунку 1. 2.2 Знак інтеграла означає, що модель є членом інтегрального оператора. Склад сигналу позначається кружком із секторами. Якщо сектор пофарбований в чорний колір, віднімiть сигнал, що йде в цей сектор (враховуйте його зі знаком «мінус» у сумі). На додаток до вже розглянутого сигналу, на рисунку показано шум вимірювання m(t), який спотворює показання датчика (у цій задачі ми будемо вважати, що шум або відсутній, або занадто малий, щоб його ігнорувати).

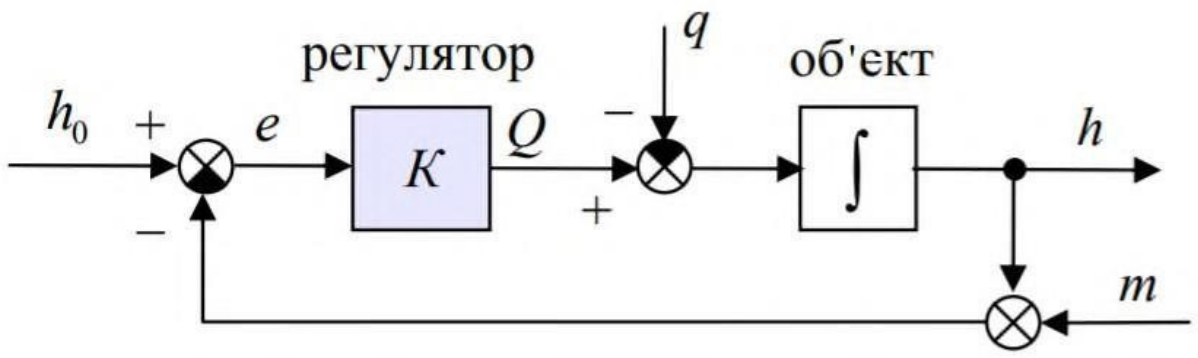


Рисунок 2.2 - Структурна схема створюваної системи управління

Щоб контролювати цю систему, ми будемо змінювати коефіцієнт K на основі поточного значення навантаження системи та подачі насоса.

Спочатку визначимо вхідні та вихідні параметри нейронної мережі. Позначимо поточну ітерацію літерою, тоді рівняння (2.6) набуває вигляду:

$$Q_i = K_i * [h_0 - h_i]. \quad (2.8)$$

Для спрощення керування ми визначаємо значення подачі насоса як суму значення подачі з попередньої ітерації та значення дельта:

$$Q_i = Q_{i-1} + K_i * [h_0 - h_i], \quad (2.9)$$

де Qi - запитуване значення подачі насоса в поточній ітерації;

Qi−1 – значення, яке надав насос на попередній ітерації;

*h*0 – бажана висота рівня води;

*h*i - висота рівня води поточної ітерації;

Знайдемо значення коефіцієнта К з цього виразу:

$$K = \frac{Q_i - Q_{i-1}}{h_0 - h_i}. \quad (2.10)$$

Продовжуйте споживати через дві секунди, можна припустити

$$Q_{i-1} = q + (h_0 - h_{i-1}), \quad (2.11)$$

$$Q_i = q + (h_0 - h_i). \quad (2.12)$$

Після підстановки рівнянь (2.10) і (2.11) у рівняння (2.9) одержимо такі вирази:

$$K = \frac{q + (h_0 - h_i) - [q + (h_0 - h_{i-1})]}{(h_0 - h_i)} = \frac{h - 2 * h_i + h_{i-1}}{(h_0 - h_i)}. \quad (2.13)$$

Як видно з рівняння (2.13), в даному випадку конкретне значення витрати води і попереднє значення подачі насоса не має значення, оскільки для даної задачі достатньо визначити збільшення подачі насоса в порівнянні з попереднім значення .

Щоб перевірити цю формулу, давайте підключимо фактичні дані до формули та обчислимо довільне значення K. Візьмемо $h0 = 1$ м, $hi{-}1 = 0{,}8$ м і $hi = 0{,}5$ м.

У цьому випадку, підставивши дані у формулу (2.13), отримаємо:

$$K = \frac{1 - 2 * 0.5 + 0.8}{(1 - 0.5)} = 1.6.$$

Перевіримо правильність визначених коефіцієнтів. З цією метою ми приймемо

Довільна витрата води, наприклад q = 0,4 м3/с. Враховуючи, що рівень води впав на 0,3 м на одній ітерації, подачу насоса на попередній ітерації можна розрахувати як різницю між витратою та різницею висоти стовпа води на поточній і попередній ітераціях.:

$$Q_{i-1} = 0.4 - (0.8 - 0.5) = 0.1.$$

Подача насоса, яку необхідно встановити, розраховується за допомогою рівняння (2.8):

$$Q_i = 0.1 + 1.6 * [1 - 0.5] = 0.9.$$

Різниця у висоті стовпа води в резервуарі розраховується як різниця між подачею і витратою води за одну секунду.:

$$\Delta h = Q_i - q = 0.9 - 0.4 = 0.5.$$

Отже, висота стовпа води в резервуарі в наступній ітерації буде дорівнює сумі різниці між висотою стовпа води в поточній ітерації та висотою стовпа води в наступній ітерації.:

$$h_{i+1} = h_i + \Delta h = 0.5 + 0.5 = 1.$$

Як бачите, наступна ітерація досягне потрібної висоти стовпа води в баку. Цей метод визначення очікуваних результатів задовольняє місію.

Важливо відзначити, що використання цього регулятора не приведе вас до положення рівноваги. При використанні різних типів регуляторів точність регулювання може бути вищою з меншою статичною похибкою або без неї.

Після визначення завдання необхідно визначити параметри нейронної мережі, які допоможуть визначити регулятор системи, математична модель якої описується рівняннями (2.1 – 2.13).

2.2 Визначення параметрів нейронної мережі

2.2.1 Визначення архітектури нейромережі

Конфігурації штучних нейронних мереж надзвичайно різноманітні. Тим не менш, мережеві парадигми мають багато спільного [23]. Штучні нейронні мережі розрізняють за топологічними типами, заснованими на структурі зв'язків між нейронами в мережі. Для вирішення різних завдань використовуються різні топології нейронної мережі.

Мережні архітектури та завдання, які їх використовують, а також те, як вони навчаються, показано в таблиці 2.1.

Таблиця 2.1 - Основні архітектури нейромереж та області застосування

Архітектура	Алгоритм навчання	Задача
Одношаровий та багатошаровий персептрон	Алгоритм навчання персептрона. Зворотне розповсюдження. *Adaline* та *Madaline*	Класифікація образів. Апроксимація функцій. Прогнозування. Керування
Рекурентна	Алгоритм навчання Больцмана	Класифікація образів
Багатошарова прямого розповсюдження	Лінійний дискримінантний аналіз	Аналіз даних. Класифікація образів
Змагання	Векторне квантування	Категоризація всередині класа. Стиснення даних
Мережа АРТ	*ARTMap*	Класифікація образів
Мережа Хопфілда	Навчання асоціативної пам'яті	Асоціативна пам'ять
Мережа Кохонена	*SON* Кохонена	Категоризація, аналіз даних
Мережа радіально базисної функції	Алгоритм навчання радіально базисної функції	Класифікація образів. Апроксимація функцій. Прогнозування. Керування

Насправді завдання полягає в апроксимації деякої функції. Для апроксимації функцій добре зарекомендувала себе штучна нейронна мережа прямого поширення - це нейронна мережа, в якій сигнал поширюється в одному напрямку, починаючи від вхідного шару нейрона, через прихований шар до вихідного шару, Результат обробки сигналу виходить на вихідному нейроні. У цьому типі мережі відсутні зворотні зв’язки.

Виберемо таку нейронну мережу для реалізації задачі визначення параметрів регулятора.

2.2.2 Визначення кількості і вмісту шарів нейромережі

Нейронна мережа складається з шарів – вхідного, вихідного та прихованого. Потрібні два шари - вхідний і вихідний. Кількість прихованих шарів може бути довільною. Кількість обчислювальних рівнів і кількість нейронів на шар не можуть бути розв’язані, і їх потрібно використовувати в штучних нейронних мережах для вирішення конкретних проблем прогнозного моделювання. Немає загального правила щодо кількості таких шарів, зазвичай вибирають від одного до трьох прихованих шарів. Чим більше нелінійна проблема, тим більше повинно бути прихованих шарів. Щоб вибрати кількість прихованих шарів, ви можете скористатися одним із запропонованих нижче методів.

- Експериментально - кількість шарів було обрано за допомогою систематичних експериментів з різною кількістю шарів і нейронів на одному наборі навчальних даних. За результатами експерименту можна вибрати відповідну кількість шарів відповідно до швидкості навчання, тенденції до повторного навчання, мінімальної похибки розрахунку тощо;

- Інтуїтивно зрозумілий - параметри ШНМ можна налаштувати відповідно до інтуїції. Наприклад, на основі розуміння предметної області можна стверджувати, що глибокі ієрархічні моделі необхідні для вирішення проблем прогнозування, або існує позитивний досвід використання ієрархічних моделей для вирішення подібних проблем. У цьому випадку необхідно вибрати конфігурацію ШНМ з кількома рівнями глибини;

- Завжди використовуйте багато шарів. Існує гіпотеза, що використання глибоких штучних нейронних мереж з великою кількістю шарів може використовуватися як евристика для налаштування мережі для вирішення завдань прогнозного моделювання. Недоліком цього методу є те, що чим більше кількість шарів нейронної мережі, тим повільніше навчання, і тим більші навчальні вибірки необхідні для навчання;

- Вчіться на попередньому досвіді. Метод полягає в пошуку реалізованих нейронних мереж для подібних завдань і на підставі результатів і деякого показника якості таких нейронних мереж вибору відповідної кількості шарів;

- Використовуйте всі перераховані методи. Ви можете знайти реалізації нейронної мережі для подібних завдань і вибрати кількість шарів і нейронів на основі конфігурації розробленої нейронної мережі. Після вивчення штучної нейронної мережі кількість шарів і нейронів у мережі можна збільшити або зменшити залежно від певних показників якості. Тобто, якщо значення помилки прийнятне, а швидкість навчання повільна, то кількість шарів можна зменшити.

Для нейронної мережі, яку ми проектуємо, давайте встановимо кількість прихованих шарів на 3. У більшості випадків цього має бути достатньо.

У ШНМ прямого розповсюдження всі елементи попереднього рівня з’єднані з усіма елементами наступного рівня. Кількість нейронів у першому та останньому шарах залежить від того, скільки полів призначено як входи та виходи. За формулою (2.13), вхідними даними проекційної нейронної мережі будуть параметри h, h_i і h_{i-1}, а вихідним значенням буде K, коефіцієнт цієїітерації. Тобто вхідний рівень нейронної мережі складається з трьох нейронів, авихідний – з одного нейрона.

Немає явного алгоритму для визначення кількості нейронів у прихованому шарі нейронної мережі. Їх не повинно бути дуже мало, тому що в цьому випадку нейронна мережа буде погано навчатися, але їх не повинно бути занадто багато, тому що навчання такої мережі може зайняти багато часу, і можливо, що деякі з них будуть використані результати обчислення без нейронної мережі. Крім того, кількість зв'язків між нейронами має бути меншою за кількість прикладів у навчальній вибірці. В іншому випадку нейронна мережа втратить здатність до узагальнення і просто «запам’ятає» всі приклади в навчальній вибірці. Потім при перевірці на прикладах у навчальній вибірці він показує відмінні результати, але на реальних даних - не дуже.

Більшу кількість нейронів можна вибрати, а потім поступово зменшувати, доки бажана точність не буде досягнута з мінімальною кількістю нейронів. Виберемо кількість нейронів у першому шарі – 60, у другому – 40, у третьому – 20.

2.2.3 Визначення методу навчання нейронної мережі

Мережа навчена генерувати бажаний набір виходів Y для набору входів X. Кожен набір таких входів (або виходів) розглядається як вектор. Навчання відбувається шляхом застосування набору значень функції до входу, визначення вихідного значення та певним чином регулюючи ваги зв'язків між нейронами.

Навчання нейронної мережі можна сформулювати як задачу оптимізації. Оцінювати означає кількісно вказати, чи добре мережа вирішує поставленеперед нею завдання. Для цього будується функція оцінювання. Загалом, цезалежить очевидно від вихідного сигналу мережі та неявно (через його функцію) від усіх її параметрів. Найпростішим і найпоширенішим прикладом оцінки є сума квадратів відстаней вихідного сигналу мережі від його бажаного значення. У цьому документі середня квадратична помилка буде використана як функція оцінки.

Існує три загальні парадигми навчання: «з учителем», «без викладача» (самонавчання) і змішане навчання.

У першому випадку нейронна мережа має правильну відповідь (вихід мережі) для кожного вхідного прикладу. Налаштуйте вагові коефіцієнти так, щоб мережа видала відповідь, яка була б якомога ближчою до відомої правильної відповіді.

Навчання «без учителя» не потребує знання правильної відповіді на кожен приклад навчальної вибірки. У цьому випадку виявляються кореляції між внутрішньою структурою даних і вибірками в навчальному наборі, щодозволяє розподіляти вибірки за класами.

У змішаному навчанні деякі ваги визначаються навчанням викладача,тоді як інші визначаються самонавчанням.

У цій роботі використовується парадигма навчання «з учителем», тобто генерується певна кількість навчальних даних, а значення ваг у зв'язках коригуються шляхом порівняння очікуваних результатів з отриманими результатами між нейронами.

Щоб змінити ваги нейронних входів, необхідно застосувати один із цих методів навчання. Одним із найпоширеніших методів навчання є неправильний метод зворотного поширення (рис. 2.3).

Це ітераційний градієнтний алгоритм для мінімізації помилок ШНМ і отримання бажаного результату. Основна ідея цього методу полягає в поширенні сигналу помилки від виходу мережі до її входу в напрямку, протилежному прямому поширенню сигналу в нормальному режимі роботи.

Алгоритм зворотного поширення помилок використовується в багатошарових персептронах. Для можливості застосування методу зворотного поширення помилок необхідно розрізняти функції активації нейронів.

Ваги нейронних входів змінюються відповідно до правила «дельта», яке є одним із найбільш широко використовуваних. Це правило засноване на простій ідеї постійної зміни зведених ваг, щоб зменшити різницю («дельта») між очікуваним і фактичним значеннями вихідного сигналу нейрона.

Відповідно до цього правила середньоквадратична помилка мережі мінімізована. Це правило також відоме як правило навчання Відроу-Гоффа та правило навчання найменшого середнього квадрата.

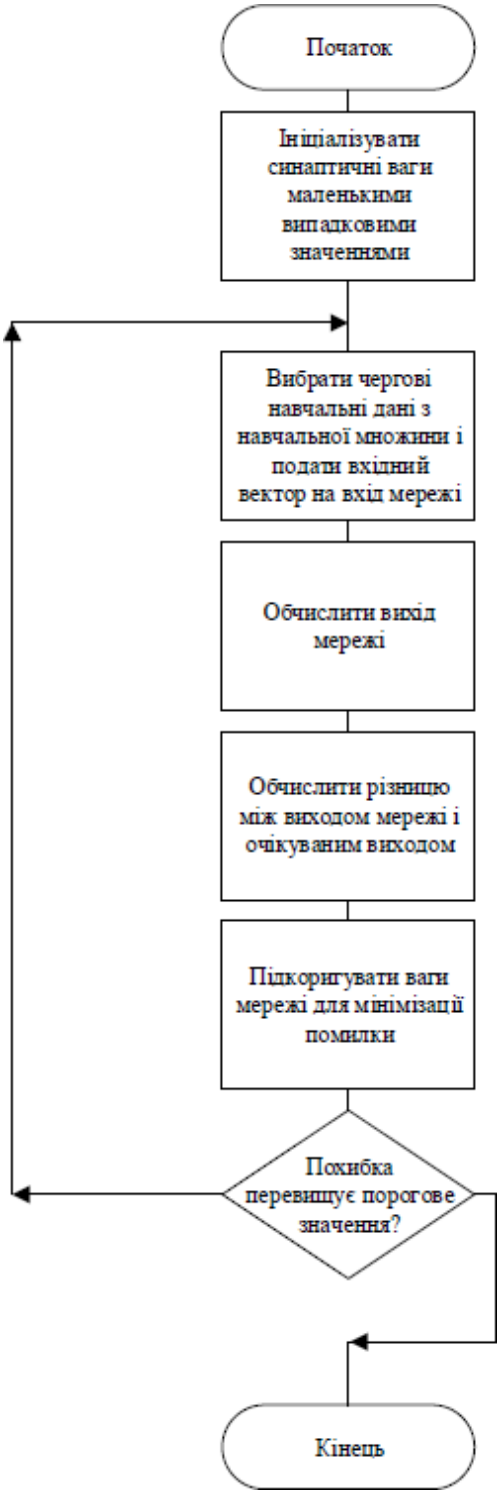


Рисунок 2.3 - Схема алгоритма зворотного поширення помилки

У правилі «дельта» помилка, отримана початковим шаром, перетворюється на похідну передатної функції та поширюється шар за шаром назад до попередніх шарів, щоб виправити ваги погоди. Процес зворотного розповсюдження мережевих помилок триває, доки не буде досягнуто першого рівня.

Хоча алгоритм зворотного поширення помилок має багато успішних застосувань, він не є панацеєю. Найнеприємнішим є нескінченно довгий процес навчання. У складних завданнях для навчання мережі можуть знадобитися дні або навіть тижні, або вона може не навчатися взагалі. Основними причинами такої поведінки є:

- параліч мережі (в процесі навчання мережі значення ваг можуть стати дуже великими через корекції, що може призвести до того, що всі або більшість нейронів будуть функціонувати при початкових значеннях дуже великі, у похідній функції дуже мала область, тому процес навчання може фактично зависнути);

– локальні мінімуми (оскільки метод зворотного розповсюдження помилок використовує свого роду градієнтний спуск, у якийсь момент мережа може застрягти в локальних мінімумах і опуститися туди, навіть якщо є глибші мінімуми);

– Розмір кроку (якщо розмір кроку фіксований і дуже малий, конвергенція відбувається надто повільно, якщо розмір кроку фіксований і занадто великий, може виникнути параліч або постійна нестабільність).

2.2.4 Визначення функцій узагальнення та активації нейронів

У наявних пакетах програмного забезпечення штучні нейрони називаються «елементами обробки» і мають набагато більше можливостей, ніж прості штучні нейрони. на рис. На рисунку 2.4 показана детальна схема спрощеного штучного нейрона.

Змінений вхід передається до функції sum, яка в основному просто підсумовує добутки. Однак можна вибрати багато різних операцій, наприклад середнє, максимальне, мінімальне, АБО, І тощо, які можуть давати будь-яку кількість різних значень. Крім того, більшість комерційних програм дозволяють розробникам програмного забезпечення створювати власні функції підсумовування за допомогою процедур, закодованих мовами високого рівня

(C, C++, Python). Іноді функція підсумовування ускладнюється шляхом додавання функції активації, яка дозволяє функції підсумовування працювати з часом.

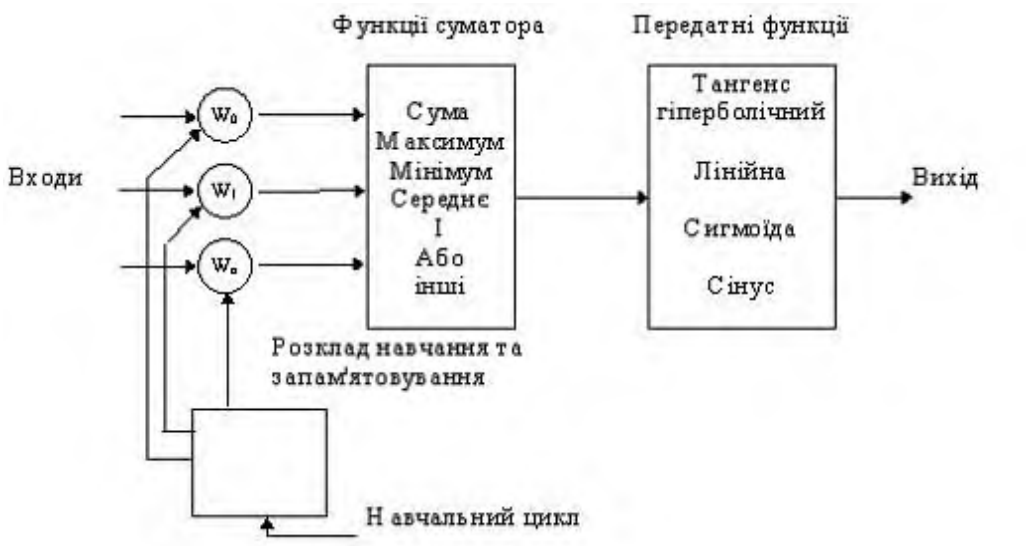


Рисунок 2.4 - Модель «елемента обробки»

Нейронна мережа, що розробляється, використовуватиме функцію підсумовування, яка є найпоширенішою функцією для нейронних мереж прямого поширення.

Для кожного шару необхідно вказати тип функції активації. Функція активації (передаточна функція) – це залежність вихідного сигналу штучного нейрона від вхідного.

Метод зворотного поширення помилок вимагає, щоб функція активації була:

- Нелінійність. Коли функція передачі є нелінійною, було показано, що двошарова нейронна мережа є загальним наближенням функції. Та ж передатна функція не має цієї властивості. Коли кілька рівнів використовують ту саму функцію передачі, вся мережа еквівалентна однорівневій моделі;

– безперервно диференційований – ця властивість бажана для використання методів оптимізації на основі градієнта;

- монотонний.

Вхідний сигнал (NET) зазвичай перетворюється за допомогою функції активації F і дає вихідний нейронний сигнал OUT = F(NET) [20]. Функція активації F(NET) може бути:

1. Порогова двійкова функція (рисунок 2.5).

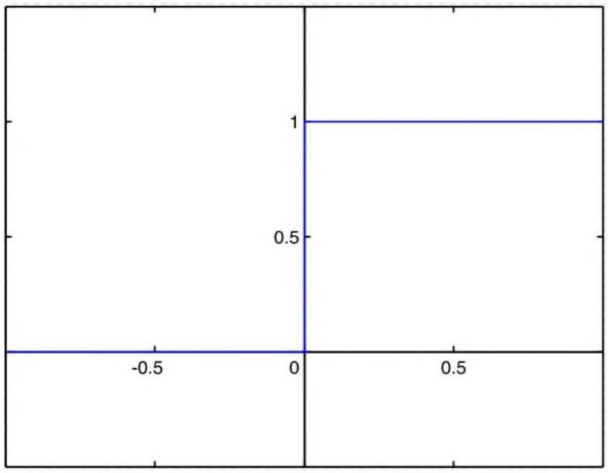


Рисунок 2.5 - Приклад графіку порогової двійкової функції

Поки зважений сигнал на вході нейрона не досягає певного порогового рівня T - вихідний сигнал дорівнює нулю. Як тільки сигнал на вході нейрона перевищує заданий рівень, вихідний сигнал стрибає. Власне, перший представник багатошарової штучної нейронної мережі – перцептрон – повністю складався з нейронів цього типу.

$$OUT = \begin{Bmatrix} 1, NET \geq T \\ 0, NET < T \end{Bmatrix}, \quad (2.13)$$

де T є деяким постійним порогом або функцією, яка більш точно моделює властивості нелінійної передачі біологічних нейронів.

2. Передатна лінійна гранична функція (рисунок 2.6).

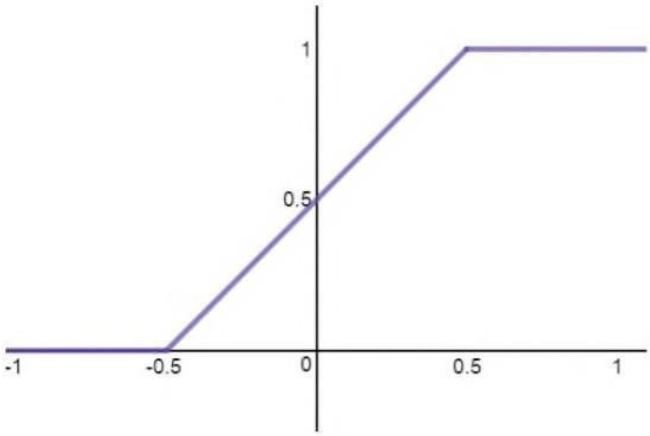


Рисунок 2.6 - Приклад графіку лінійної функції

Тут є дві лінійні частини, де функція активації точно дорівнює максимальним і мінімальним допустимим значенням, і одна, де функція строго монотонно зростає

$$OUT = \begin{Bmatrix} 1, NET \geq T \\ NET, o \leq NET < T \\ 0, NET < 0 \end{Bmatrix}. \quad (2.14)$$

3. Сігмоїдна (S-подібна) або логістична функція (рис. 2.7).

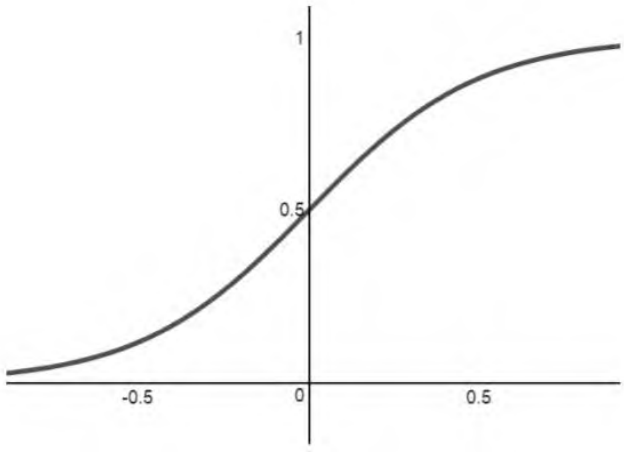


Рисунок 2.7 - Приклад графіку сігмоїдної функції

Обмежені можливості ШНМ з пороговими функціями активації нейронів призводять до використання функцій сигмоїдного типу. У той же час активація нейронів відбувається плавно, дозволяючи перемикатися з двійкового виходу на аналоговий. Особливістю сигмоїдних функцій є те, що вони підсилюють слабкі сигнали, але не насичуються сильними сигналами.

$$OUT = \frac{1}{1 + e^{-C*NET}}. \tag{2.15}$$

4. Функцією гіперболічного тангенсу (рис. 2.8).

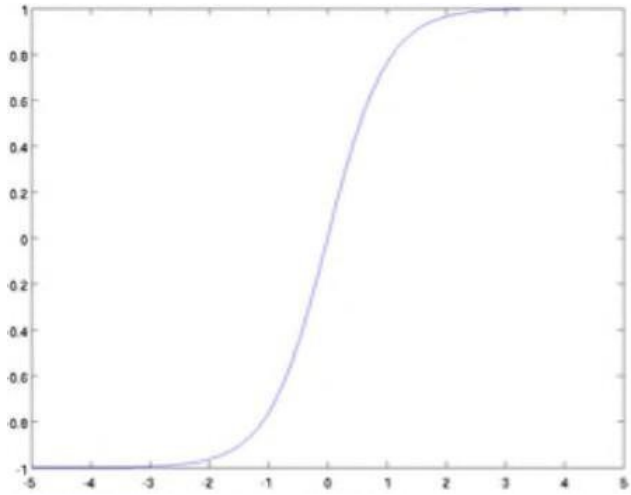


Рисунок 2.8 - Приклад графіка гіперболічного тангенсу

Гіперболічний тангенс дуже схожий на сигмоподібну, оскільки є модифікованою сигмоподібною функцією. Отже, така функція має ті ж властивості, що й сигма. Його нелінійний характер робить його ідеальним для об’єднання шарів, а функція важливості коливається від мінус одиниці до одиниці. Тобто функція активації не буде перевантажена великими значеннями. Однак варто зауважити, що градієнт функції тангенса більший, ніж градієнт сигмоїди (похідна має більший нахил). Рішення про те, чи використовувати сигмоподібну чи дотичну, має залежати від вимог до величини градієнта.

$$OUT = th(C * NET). \quad (2.16)$$

5. Лінійною (обмеденою) функцією (рис. 2.9).

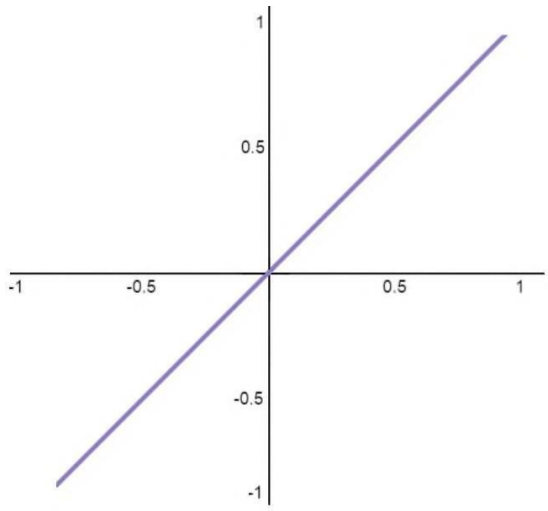


Рисунок 2.9 - Приклад лінійної обмеденої функції

Вихідний сигнал нейрона лінійно залежить від зваженої суми його вхідного сигналу. Іншими словами, на виході нейрона з цією функцією активації ми отримуємо вхід на його вхід. У ШМН з кількома рівнями нейрониз заданою функцією активації зазвичай використовуються для вихідного або вхідного рівнів.

Такі функції активації використовуються рідко, оскільки ШНМ, що містять лише нейрони з лінійними функціями активації, можуть виконувати дуже обмежений набір операцій (зміна знака вхідних даних, підсумовування тощо).

6. Випрямленою лінійною, або ReLU – rectified linear unit (рис. 2.10).

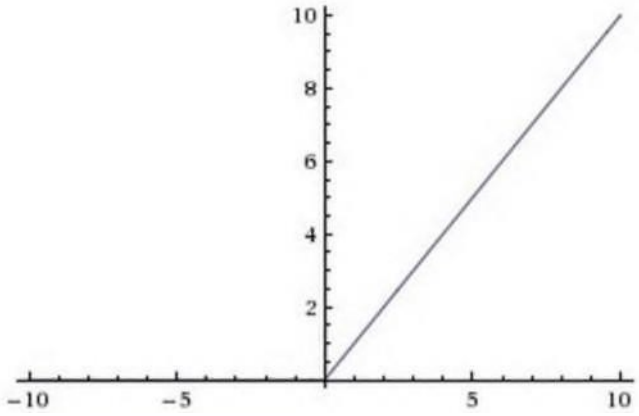


Рисунок 2.10 - Приклад випрямленої лінійної функції

Важливою перевагою цієї функції над сигмою та гіперболічним тангенсом є те, що обчислення значення функції є менш дорогим і, отже, швидшим.

$$OUT = \begin{Bmatrix} NET, NET \geq 0 \\ 0, NET < 0 \end{Bmatrix}. \quad (2.17)$$

Існує багато різних варіантів наведених вище функцій активації, кожна з яких має застосування в певних областях.

Вибір конкретної функції активації залежить від функції апроксимації та її визначення. Наприклад, для визначення значення функції синуса зазвичай вибирають сигмоподібну функцію або гіперболічну функцію тангенса, оскільки вони дозволяють меншій кількості нейронів у шарі отримати вищу точність нейронної мережі.

Для нейронної мережі, що розробляється, використовуватиметься параметрична випрямлена лінійна функція (PReLu), оскільки вона дає хороші результати, але займає менше часу для обчислення похідної (яка необхідна для навчання нейронної мережі, на відміну від сигмоїдної та дотичної)) і оцінки результат функції. PReLu є варіантом функції ReLu.

ReLu за своєю суттю нелінійний, і комбінація ReLu також нелінійна. Така функція є хорошим апроксиматором, оскільки будь-яка функція може бути

апроксимована композицією ReLu. Діапазон допустимих значень для ReLu - від нуля до нескінченності, тобто активація може «вибухнути».

Наступний момент – розрідженість активації. Під час роботи з великими нейронними мережами з багатьма нейронами використання сигмоподібного або гіперболічного тангенсу потребує аналогічної активації всіх нейронів. Це означає, що майже всі активації повинні бути оброблені для опису мережевого виходу. Іншими словами, активації щільні, що дорого.

В ідеалі деякі нейрони не повинні бути активовані, що робить активації рідкісними та ефективними. ReLu дозволяє це зробити. Припустимо, що існує мережа з випадково ініціалізованими вагами (або нормалізацією), де близько 50% активацій близькі до нуля через властивість ReLu (повертає 0,1 для від'ємних значень x). У такій мережі, що містить меншу кількість нейронів (розріджена активація), сама мережа стає легшою [19].

Однак ця функція активації також має свої недоліки. Оскільки частина ReLu є горизонтальною лінією (для від'ємних значень X), градієнт на цій частині дорівнює 0. Оскільки градієнт дорівнює нулю, ваги не коригуються під час спуску. Це означає, що нейрони в цьому стані не реагуватимуть на помилки/зміни введення (тільки тому, що градієнт дорівнює нулю, нічого не зміниться). Це явище називається проблемою Dying ReLu. Через цю проблему деякі нейрони просто вимикаються і не реагують, залишаючи значну частину нейронної мережі в пасивному стані.

Однак деякі варіанти ReLu допомагають уникнути цієї проблеми. Наприклад, має сенс замінити горизонтальну частину функції на лінійну. Якщо вираз лінійної функції для області $x < 0$ задано виразом $y = 0{,}01x$, лінія буде трохи зміщена від горизонтального положення. Існують інші способи уникнути нульових градієнтів. Основна ідея тут полягає в тому, щоб зробити градієнт ненульовим і поступово відновлюватися під час тренувань.

Нейронна мережа буде використовувати PReLu з параметром 0,01.Тому в другому розділі роботи розглядається перше завдання, для вирішення якого необхідно розробити нейромережевий додаток для визначення параметрів

авторегулятора. Розглянемо формули, за допомогою яких можна створити набір навчальних даних. З огляду на існуючі топології ШНМ, перераховані існуючі методи навчання для різних топологій, типи функцій активації та підсумовування, а також типи нейронів. Після аналізу наведених даних вибирається топологія та основні параметри штучної нейронної мережі, такі як: функція суматора, функція активації, метод навчання, кількість шарів і нейронів штучної нейронної мережі.

3 РОЗРОБКА НЕЙРОМЕРЕЖЕВОГО ЗАСТОСУНКУ ДЛЯ РОЗРАХУНКУ ПАРАМЕТРІВ АВТОМАТИЧНОГО РЕГУЛЯТОРА

3.1 Розробка програмного коду

3.1.1 Створення користувацького інтерфейсу

Програма розроблена для створення додатків нейронних мереж, тому інтерфейс користувача може бути простим. Основне завдання цього інтерфейсу – надати користувачеві можливість вибирати конфігурацію ШНМ, мати можливість змінювати налаштування параметрів навчання нейронної мережі, а також надавати можливість додавати файли з навчальними і тестовими зразками даних.

Вся взаємодія з користувачем відбувається в командному рядку (рис. 3.1).

```
Choose the action:
1) Initialize existing Neural Network
2) Create new Neural Network
3) Exit
```

Рисунок 3.1 - Опції командного рядка користувацького інтерфейса

Тут користувач може вибрати один із варіантів: ініціалізувати існуючу нейронну мережу, раніше збережену в окремому файлі, створити нову нейронну мережу або завершити роботу програми.

За цей вибір відповідає метод getNeuralNetwork.

```
public NeuralNetwork getNeuralNetwork() throws IOException, ClassNotFoundException {
System.out.println("Choose the action: \n1) Initialize existing Neural Network \n2) Create new Neural Network \n3) Exit");
String option = reader.readLine();
```

```
50
switch (option) {
case "1":
return this.getExistingNetwork();
case "2":
return this.createNewNetwork();
default:
return null;
}
}
```

Залежно від вибору користувача, виклик відповідного методу - ініціалізація нейронної мережі з файлу в методі getExistingNetwork або створення нової нейронної мережі в методі createNewNetwork.

```
private NeuralNetwork getExistingNetwork() throws ClassNotFoundException, IOException {
NeuralNetwork network;
System.out.println("Enter the name of the file with Neural Network: ");
String fileName = reader.readLine();
FileInputStream fis = new FileInputStream(fileName);
ObjectInputStream oin = new ObjectInputStream(fis);
network = (NeuralNetwork) oin.readObject();
oin.close();
fis.close();
return network;
}
private NeuralNetwork createNewNetwork() throws IOException {
int numOfLayers;
while (true) {
```

```
System.out.println("Enter the total number of layers, including input and output layers: ");
numOfLayers = Integer.parseInt(reader.readLine());
51
if (numOfLayers < 2) {
System.out.println("The network should consist of 2 layers at least!");
} else {
break;
}
}
int[] numOfNeuronsInLayers = new int[numOfLayers];
ArrayList<ActivationFunction> activationFunctions = new ArrayList<>();
for (int i = 0; i < numOfLayers; i++) {
int numOfNeurons;
while (true) {
System.out.println("Enter the number of neurons in " + (i + 1) + " layer: ");
numOfNeurons = Integer.parseInt(reader.readLine());
if (numOfNeurons < 1) {
System.out.println("The layer should consist of 1 neuron at least!");
} else {
numOfNeuronsInLayers[i] = numOfNeurons;
break;
}
}
if (i != 0) {
activationFunctions.add(this.getActivationFunction(reader, i));
}
}
return new NeuralNetwork(numOfNeuronsInLayers, activationFunctions);
}
```

Після ініціалізації нейронної мережі з файлу або створення ШНМ з необхідними параметрами користувачеві пропонується вибрати один із таких варіантів (рис. 3.2):

– Розпочати навчання нейронної мережі (процес навчання та логіка будуть представлені пізніше);

– Виконати розрахунок (з’явиться запит на введення вхідних даних, після завершення обчислення результат розрахунку буде виведено на консоль);

– зберегти нейронну мережу у файл;

– вибрати іншу нейронну мережу (користувач повернеться до попереднього вибору);

– Змінити налаштування, або вивести інформацію про нейронну мережу в консоль (рис. 3.3).

```
Press:
1 - to train the network
2 - to process data
3 - to save the network
4 - to choose another network
5 - to change train settings
6 - to print neural network in console
Any other key - to exit
```

Рисунок 3.2 - Доступні опції для налаштування нейронної мережі

Після виведення інформації ШНМ на консоль користувач може переглянути кількість шарів, кількість нейронів у шарах, їх функції активації та ваги зв’язків між нейронами.

Залежно від обраних параметрів взаємодія користувача з нейронною мережею описується в методі handleUserInteraction.

```
Press:
1 - to train the network
2 - to process data
3 - to save the network
4 - to choose another network
5 - to change train settings
6 - to print neural network in console
Any other key - to exit

6
Neural network: {
    Layer #1 of 2:  {
        Neuron #1 of 2: {
            activation function: ActivationFunctions.Identity
            weights:
        }
        Neuron #2 of 2: {
            activation function: ActivationFunctions.Identity
            weights:
        }
    }
    Layer #2 of 2:  {
        Neuron #1 of 1: {
            activation function: ActivationFunctions.Sigmoid
            weights: 0.01380113 -0.24846941
        }
    }
}
```

Рисунок 3.3 - Виведення інформації про нейромережу в консоль

```
public NeuralNetwork handleUserInteraction(NeuralNetwork
network) throws IOException, ClassNotFoundException {
System.out.println("Press:\n" +
"1 - to train the network\n" + "2 - to process data\n" +
"3 - to save the network\n" +
```

```
"4 - to choose another network\n" + "5 - to change train settings\n" +
"6 - to print neural network in console\n" + "Any other key - to exit\n");
String option = reader.readLine(); switch (option) {
case "1":
return this.trainNeuralNetwork(network); case "2":
return this.processData(network); case "3":
return this.saveNetwork(network); case "4":
this.trainFileName = null; this.verificationFileName = null; return this.getNeuralNetwork();
case "5":
this.changeTrainSettings(reader); return network;
case "6":
System.out.println(network); return network;
default:
return null;

}
}
```

Вибравши опцію зміни параметрів навчання ШНМ, користувач може змінити наступні параметри (рис. 3.4):

– змінити розмір кроку помилки (кількість, на яку буде помножена помилка нейронної мережі при обчисленні ваг з’єднання нейронів);

– Змініть інтервал епох між перевірками правильності навчання ШНМ на тестовому наборі даних (інтерфейс користувача зчитує файл із тестовими зразками, надсилає вхідні дані на вхід нейронної мережі, порівнює вихідні дані з очікуваним значенням, обчислює середній квадратний корінь помилка, виведення на консоль після розрахунку);

– зміна кількості епох (кількості ітерацій навчання нейронної мережі, де одна ітерація відповідає одному проходу через весь набір даних навчальних вибірок);

– змінити допустиму похибку (якщо середньоквадратична похибка дорівнює цьому значенню, нейронна мережа припиняє навчання);

– Змінити період (в епохах), який ШНМ автоматично зберігає у файлі (при виникненні помилки час, витрачений на навчання нейронної мережі, не витрачається даремно);

- Зміна періодичності (по епохах) для видалення зв'язків і нейронів, які не беруть участь у формуванні кінцевого результату.

```
Press:
1 - to change the Penalty Step (1.0E-11)
2 - to change the Check Error Period (in epoch unit) (10000)
3 - to change Number Of Epoch (1000000000)
4 - to change Allowable Error (1.0E-5)
5 - to change Auto Save Period (in epoch unit) (100000)
6 - to change the Delete Links Period (in epoch period) (500000)
Any other key to return to the previous menu
```

Рисунок 3.4 - Опції управління навчанням нейромережі

3.1.2 Структура і вміст шарів нейромережі

На основі інформації попередніх розділів необхідно розробити програмний код для навчання та обробки даних штучних нейронних мереж. Відповідно до принципів об'єктно-орієнтованого програмування хорошим рішенням є розділення реалізації програми на кілька модулів. Основним модулем є нейронна мережа, яка представлена окремим класом на мові програмування Java. Він містить інформацію про шар, кількість нейронів у шаріта функцію активації в шарі. При ініціалізації нейронної мережі дані передаються в контролер класу, де ініціалізуються шари.

Кожен шар містить посилання на нейрони попереднього шару (крім першого шару), власні нейрони та функцію активації.

Залежно від того, чи є шар вхідним, називається одна з функцій - initInputNeurons або initNeurons, які ініціалізують вхідні нейрони, вихідні нейрони або нейрони прихованого шару.

```
public void initInputNeurons(ArrayList<FloatDataWrapper> inputValues) {
neurons = new ArrayList<>();
for(FloatDataWrapper singleValue : inputValues) { neurons.add(new
InputNeuron(singleValue, activationFunction));
}
}
public void initNeurons(int numOfNeurons, Boolean isOutputLayer) { neurons
= new ArrayList<>();
for(int i = 0; i < numOfNeurons; i++) {
neurons.add(isOutputLayer ? new OutputNeuron(inputNeurons,
activationFunction) : new Neuron(inputNeurons, activationFunction));
}
}
```

Вхідні та вихідні нейрони є різними випадками реалізації нейронів.

Важливою відмінністю вхідного нейрона є те, що він не має суматорів і функцій активації. Тобто на виході такого нейрона буде те, що передається на вхід нейрона.

```
public float  getCalculatedOutput() { this.outputData = inputData.getData();
return this.outputData;
}
```

Вихідний нейрон, на відміну від вхідного, навпаки – має вхідні зв’язки і саме з нього починається обрахування похибки.

```
public void adjustPenalties() { for(Link singleLink : inputLinks) {
singleLink.adjustPenalty(this.penalty * Math.abs(singleLink.getWeight() / sumOfWeights));
}
}
```

Нейрони в прихованому шарі однакові, тільки кількість вхідних з'єднань і функції активації різні. Вхідне значення такого нейрона обчислюється як сума значень вхідних нейронів, помножених на відповідну вагу вхідного з'єднання (реалізація суматора).

```
protected void initInputData() { this.inputData = 0.0f; this.sumOfWeights = 0.0f; for(Link singleLink : inputLinks) {
this.inputData += singleLink.getOutputData(); sumOfWeights += Math.abs(singleLink.getWeight());
}
}
```

Обчисліть вихідне значення нейрона відповідно до обраної функції активації.

```
public float getCalculatedOutput() { this.initInputData(); this.setPenalty(0.0f);
this.outputData = function.calculate(this.inputData); return this.outputData;
}
```

3.1.3 Реалізація методу зворотного поширення похибки

На додаток до функції, зазначеної в попередньому підрозділі, нейрон містить функцію, яка обчислює помилку та відповідно коригує ваги.

```
public void adjustPenalties() {
float   finalPenalty =       this.penalty *
function.getDerivative(this.inputData, this.outputData);
for(Link singleLink : inputLinks) { singleLink.adjustPenalty(finalPenalty);
}
}
```

Метод зворотного поширення помилок використовує похідну функції активації для визначення значення, на яке слід зменшити або збільшити вагу з’єднання.

```
public void adjustPenalties() {
float   finalPenalty =       this.penalty *
function.getDerivative(this.inputData, this.outputData);
for(Link singleLink : inputLinks) { singleLink.adjustPenalty(finalPenalty);
}
}
```

Це значення залежить від того, наскільки вихідне значення даного нейрона впливає на вихід нейронної мережі. Чим більший вплив певного нейрона, тим більше значення помилки для цього нейрона.

На підставі помилки конкретного нейрона обчислюється помилка вхідного нейрона. Крім того, корекція відбувається на кожному вхідному нейроні, тобто на нейронах попереднього шару. Існує ймовірність того, що в якийсь момент часу вага з'єднання може стати дуже великим або дуже малим

значенням (або навіть нескінченністю за межами допустимого діапазону значень для чисел з плаваючою комою). Щоб вирішити цю проблему, вага цього з'єднання ініціалізується до випадкового значення.

```
public void adjustPenalty(float penalty) {
if (Float.isInfinite(this.outputData) ||
Float.isNaN(this.outputData) || this.weight < -1E20 || this.weight > 1E20) {
this.weight = (float) (0.5 - Math.random() * 1);
} else if (!isTriggered) { needToBeDeleted = true;
} else {
if(Float.isInfinite(penalty) || Float.isNaN(penalty)) { return;
}
needToBeDeleted = false; this.input.setPenalty(this.input.getPenalty() +
penalty * weight);
this.weight += NeuralNetworkSettings.PENALTY_STEP * penalty
* this.neuronOutputData;
}
}
```

Видалити підключення з нейронної мережі, які не використовувалися для обчислень протягом певного періоду часу. Нейрони з усіма видаленими вхідними зв’язками також видаляються з нейронної мережі, оскільки такі нейрони жодним чином не впливають на вихід нейронної мережі.

Швидкість зміни з'єднання визначається змінною PENALTY_STEP, яку користувач може вказати під час навчання нейронної мережі.

Ця змінна визначає розмір кроку градієнтного спуску, і якщо значення дуже велике, можуть бути випадки, коли оптимальне рішення може прослизнути за глобальний мінімум і ніколи не повернутися до нього. Якщо значення дуже мале, система може застрягти в локальних мінімумах і не досягти оптимального рішення. Рекомендується починати з невеликого

значення коефіцієнта похибки і при необхідності збільшувати його під час навчання, якщо подальше навчання не призводить до зменшення похибки.

3.1.4 Функції активації

Функція активації повинна обчислювати вихідне значення нейрона і брати участь у заданій помилці обчислення. Ці дві функції об'єднані в одному інтерфейсі ActivationFunction.

```
public interface ActivationFunction { Float calculate(Float parameter);
Float getDerivative(Float parameter, Float result);
}
```

Цей інтерфейс лише визначає, які методи мають бути реалізовані в класі функції активації. Конкретну реалізацію методу необхідно створити у відповідному класі функції активації за формулами (2.13 - 2.17).

Наприклад, спосіб реалізації функції активації PReLu:

```
public class PReLU implements ActivationFunction, Serializable {
public Float calculate(Float parameter) {
return (parameter < 0 ? 0.01f * parameter : parameter);
}
public Float getDerivative(Float parameter, Float result) { return (parameter < 0 ? 0.01f : 1);
}
```

Приклад методів для реалізації активаційної функції сігмоїди:

```
public class Sigmoid implements ActivationFunction, Serializable { public Float calculate(Float parameter) {
```

```
return (float) (1 / (1 + Math.exp(-parameter)));
}
public Float getDerivative(Float parameter, Float result) { return result * (1 - result);
}
}
```

Приклад методів реалізації ионотонної функції активації:

```
public class PReLU impublic class Identity implements ActivationFunction, Serializable { public Float calculate(Float parameter) {
return parameter;
}
public Float getDerivative(Float parameter, Float result) { return 1f;
}

}
```

Різницю в часі, необхідному кожній функції для обчислення значення та похідної, можна побачити в цих трьох прикладах – sigmoid використовує складнішу арифметику, тому обчислення значення та похідної займає більше часу. Ця ж функція активації в свою чергу повертає значення параметра, похідна якого чітко визначена і дорівнює 1, тому при використанні такої функції час на обчислення значення і похідної буде мінімальним. І хоча різницяне дуже велика, у випадку великої кількості нейронів вибір функції активації має значний вплив на швидкість навчання нейронної мережі.

3.1.5 Розробка функції навчання нейронної мережі

Основним методом визначення вихідного значення нейронної мережі є метод processData().

```
public ArrayList<FloatDataWrapper> processData() { ArrayList<FloatDataWrapper> outputDataWrappers = new ArrayList<>(); for (Neuron outputNeuron : layers.get(layers.size() - 1).getNeurons()) {
outputDataWrappers.add(new FloatDataWrapper(outputNeuron.getCalculatedOutput()));
}
return outputDataWrappers;
}
```

Цей метод викликає метод getCalculatedOutput() у нейронах вихідного шару, який, у свою чергу, рекурсивно викликає обчислення значень нейронів у попередніх шарах.

Метод train() є основним методом навчання нейронних мереж. В якості аргументу йому передається файл, з якого потрібно прочитати дані. Цей метод зчитує дані рядок за рядком, перетворює їх у придатний для використання формат і передає перетворені дані в метод trainIteration().

```
public void train(BufferedReader fileReader) throws IOException { String line;
while (true) {
line = fileReader.readLine();
if (line == null || line.isEmpty()) { break;
}
String[] parameters = line.split("\\$");
this.trainIteration(NeuralNetworkService.getDataWrappersFromString(parameters
[0]), NeuralNetworkService.getDataWrappersFromString(parameters[1]));
}
```

Метод trainIteration() отримує вхідні дані з файлу, перетворює їх у зручний формат і передає ці дані в нейронну мережу для обчислення. Після отримання результатів роботи нейронної мережі дані передаються в метод adjustPenalties(), який обчислює помилку для кожного вихідного нейрона і викликає цей метод для зміни вагових коефіцієнтів зв'язків між нейронами. Вагові коефіцієнти змінюються шар за шаром, від вихідного рівня до вхідного (якщо є кілька вихідних нейронів, помилка обчислюється шар за шаром для кожного нейрона окремо).

```
private void adjustPenalties(ArrayList<FloatDataWrapper> expectedResults, ArrayList<FloatDataWrapper> actualResults) {
ArrayList<Neuron> outputNeurons = this.layers.get(this.layers.size() - 1).getNeurons();
for (int i = 0; i < outputNeurons.size(); i++) {
float delta = expectedResults.get(i).getData() - actualResults.get(i).getData();
outputNeurons.get(i).setPenalty(delta);
}
for (int i = layers.size() - 1; i > 0; i--) {
for (Neuron outputNeuron : this.layers.get(i).getNeurons()) { outputNeuron.adjustPenalties();
}
}
}
private void trainIteration(ArrayList<FloatDataWrapper> inputData, ArrayList<FloatDataWrapper> outputData) {
this.adjustPenalties(outputData, this.getResultsByParameters(inputData));
}
```

Спосіб зміни ваг залежить від вибраної функції активації шару.

Перевірити роботу ШНМ і визначити середньоквадратичну помилку в методі getError(), який зчитує тестові дані з файлу, подає вектор вхідних даних на вхід нейронної мережі та обчислює середньоквадратичну помилку на основі отриманих даних.

```
public float getError(BufferedReader fileReader) throws IOException { ArrayList<FloatDataWrapper> actualResults = new ArrayList<>(); ArrayList<FloatDataWrapper> expectedResults = new ArrayList<>(); String line;
while (true) {
line = fileReader.readLine();
if (line == null || line.isEmpty()) { break;
}
String[] parameters = line.split("\\$"); expectedResults.addAll( NeuralNetworkService.getDataWrappersFromString(parameters[1]));
actualResults.addAll(this.getResultsByParameters( NeuralNetworkService.getDataWrappersFromString(parameters[0])
)
);
}
float error = 0f;
for (int i = 0; i < actualResults.size(); i++) { error += Math.pow(
expectedResults.get(i).getData() - actualResults.get(i).getData(), 2
);
}
return (error / expectedResults.size());
}
```

Якщо значення помилки менше допустимого значення, нейронна мережа припиняє навчання. Якщо значення помилки не змінюється між двома

перевірками помилок, нейронна мережа припиняє навчання та просить користувача змінити розмір кроку градієнтного спуску.

Окремої уваги заслуговує метод видалення зв'язків, які не беруть участь у обчисленні результату. Крім того, якщо нейрон не має вхідного з’єднання, він також вважається таким, що не впливає на вихід нейронної мережі, і його слід видалити. Метод deleteZeroLinksAndNeurons() поступово, для кожного рівня, видаляє зв’язки з нейронів, а потім виконує ітерацію по нейронах, щоб видалити ті нейрони, які не містять жодних зв’язків.

```
public void deleteZeroLinksAndNeurons() { for(int i = layers.size() - 2; i > 0; i--) {
ArrayList<Neuron> neurons = layers.get(i).getNeurons(); for(int k = 0; k < neurons.size(); k++) {
for(int j = 0; j < neurons.get(k).getInputLinks().size(); j++) { neurons.get(k).getInputLinks().removeIf(Link::isNeedToBeDeleted);
//pass the reference of the method and delete links with zero weight
}
neurons.removeIf(Neuron::isNeedToBeDeleted);    //delete if no input links

exist
}
}
}
```

Після навчання нейромережа може бути збережена в окремому файлі для подальшого використання як модуля в будь-якій іншій програмі.

Частковий перелік кодів наведено в Додатку Б.

3.2 Підготовка тестувальних даних

Підготовка тестових даних є дуже важливим етапом створення нейронної мережі. Швидкість навчання, здатність до перенавчання та здатність нейронної мережі обчислювати значення поза діапазоном значень навчальних вибірок залежать від якості та кількості навчальних вибірок.

Перш ніж почати навчання нейронної мережі, давайте згенеруємо набір тестових даних, щоб нейронна мережа могла зрозуміти, яким має бути вихід мережі. Для цього ми напишемо клас, який реалізує цю функціональність (рис. 3.5).

```
import DataWrappers.DataWrapper;
import DataWrappers.FloatDataWrapper;

import java.util.ArrayList;

public class WaterPumpFunction implements Function {

    private ArrayList<DataWrapper> inputDataWrappers;

    public ArrayList<DataWrapper> getInputData() {
        this.inputDataWrappers = new ArrayList<>();
        float h = (float) Math.round(1 + Math.random() * 4);
        float hi_1 = (float) (Math.random() * (h + 1));
        float hi = (float) (Math.random() * (h + 1));
        inputDataWrappers.add(new FloatDataWrapper(h));
        inputDataWrappers.add(new FloatDataWrapper(hi_1));
        inputDataWrappers.add(new FloatDataWrapper(hi));
        return inputDataWrappers;
    }

    @Override
    public ArrayList<DataWrapper> getOutputData() {
        ArrayList<DataWrapper> outputDataWrappers = new ArrayList<>();
        float h = ((FloatDataWrapper)inputDataWrappers.get(0)).getData();
        float hi_1 = ((FloatDataWrapper)inputDataWrappers.get(1)).getData();
        float hi = ((FloatDataWrapper)inputDataWrappers.get(2)).getData();
        float K = (h - 2 * hi + hi_1) / (h - hi);
        outputDataWrappers.add(new FloatDataWrapper(K));
        return outputDataWrappers;
    }
}
```

Рисунок 3.5 - Код класу для реалізації процесу підготовки даних

Після генерації даних за створеними класами отримують такі навчальні дані (рисунок 3.6).

```
3.0|0.06695611|2.7161705$-8.333825
4.0|1.1502964|2.379620$0.24133591
2.0|0.57117814|0.45214453$1.0769024
3.0|0.60592705|0.951815$0.8311244
4.0|2.8836582|4.64209$3.7385979
4.0|1.1257081|3.4843764$-3.5743997
3.0|2.185882|0.7752909$1.6340564
1.0|1.5482186|0.8988663$7.4223576
3.0|0.64415556|3.743722$5.1676393
4.0|2.903114|0.796401$1.6576083
1.0|0.23777987|1.731028$3.0426676
5.0|0.8194477|0.542518$1.0621266
1.0|1.1575632|1.111035$0.58096504
4.0|4.7459483|0.349078$2.2043178
5.0|4.807298|3.404119$1.8792505
2.0|2.2518208|0.5861940$2.1781156
1.0|1.2159507|1.784348$1.724675
5.0|0.15232188|0.969033$0.7973906
3.0|3.4662588|1.512024$2.3133512
1.0|1.3467082|0.1896828$2.4278667
5.0|2.5305576|4.589836$-4.0206237
3.0|3.3767183|1.694748$2.2886174
3.0|1.1100286|2.082283$-0.059427273
2.0|2.7424107|1.437705$3.3203242
3.0|0.3333523|1.142603$0.56430894
5.0|1.774804|1.415543$1.1002274
1.0|1.2022313|1.0240799$-6.3983374
4.0|3.8564098|0.599359$1.9577756
2.0|1.1048005|2.456900$3.9592862
4.0|2.4196606|4.899652$3.7566118
2.0|0.23197436|2.741820$4.3833594
5.0|4.1953235|4.503275$0.3800349
5.0|3.901594|1.395775$1.6952448
5.0|1.2689191|4.274203$-3.1406662
```

Рисунок 3.6 - Фрагмент навчальних даних

Щоб полегшити роботу з даними з комп’ютером, дані впорядковані в певній формі: «перший вхідний параметр» | «другий вхідний параметр» | ... | «останній вхідний параметр» $ «перший вихідний параметр" | "перший вхідний параметр" Два вихідних параметра" | ...

|"Останній вихідний параметр".

Кожен набір даних починається з нового рядка.

Давайте розглянемо роботу програми, яка генерує навчальні дані. Для цього можна взяти набір даних і включити його у формулу (2.12).

$$K = \frac{3 - 2 \times 2.71617 + 0.06696}{(3 - 2.71617)} = -8.33379.$$

Візьмемо довільну витрату води, скажімо q = 0,4 м3/с. Враховуючи, що рівень води впав на 0,3 м на одній ітерації, подачу насоса на попередній ітерації можна розрахувати як різницю між витратою та різницею висоти стовпа води на поточній і попередній ітераціях.:

$$Q_{i-1} = 2.71617 - (0.06696 - 0.4) = 3.04921.$$

Подача насоса, яка повинна бути встановлена, розраховується за допомогою рівняння (2.8):

$$Q_i = 3.04921 + (-8.33379 \times [3 - 2.71617]) = 0.68383.$$

Різниця у висоті стовпа води в резервуарі розраховується як різниця між подачею і витратою води за одну секунду.:

$$\Delta h = Q_i - q = 0.68383 - 0.4 = 0.28383.$$

Отже, висота стовпа води в резервуарі в наступній ітерації буде дорівнює сумі різниці між висотою стовпа води в поточній ітерації та висотою стовпа води в наступній ітерації.:

$$h_{i+1} = h_i + \Delta h = 2.71617 + 0.28383 = 3.$$

Отримано шуканий результат.

3.3 Навчання нейронної мережі

Наступним кроком після створення навчальних і тестових даних є навчання нейронної мережі. Оскільки процес визначення параметрів нейронної мережі не є формалізованим, при розробці нейронної мережі легко зробити помилки, в результаті чого нейронна мережа вчитиметься довго або взагалі не навчатиметься. .Створимо нейронну мережу з параметрами, зазначеними у другій частині статті (рис. 3.7).

Параметри навчання нейронної мережі предсствлені на рис. 3.8.

На початку навчання нейронної мережі значення середньоквадратичної похибки досить велике, але поступово, зі збільшенням кількості ітерацій і зміною кроку градієнтного спуску, досягається прийнятне значення похибки 0,01. отримано (рисунок 3.9). Подальші зміни параметрів навчання не призводять до значних змін (але кількість шарів, нейронів і функцій активації можна змінити для досягнення інших результатів), тому навчання можна припинити.

```
1) Initialize existing Neural Network
2) Create new Neural Network
3) Exit
2
Enter the total number of layers, including input and output layers:
5
Enter the number of neurons in 1 layer:
3
Enter the number of neurons in 2 layer:
60
Choose an activation function for the 2 layer:
1 - PReLU
2 - ReLU
3 - Identity
4 - Sigmoid

1
Enter the number of neurons in 3 layer:
40
Choose an activation function for the 3 layer:
1 - PReLU
2 - ReLU
3 - Identity
4 - Sigmoid

1
Enter the number of neurons in 4 layer:
20
Choose an activation function for the 4 layer:
1 - PReLU
2 - ReLU
3 - Identity
4 - Sigmoid

1
Enter the number of neurons in 5 layer:
1
```

Рисунок 3.7 - Встановлення параметрів нейронної мережі

```
Press:
1 - to change the Penalty Step (1.0E-6)
2 - to change the Check Error Period (in epoch unit) (1000)
3 - to change Number Of Epoch (500000)
4 - to change Allowable Error (1.0E-4)
5 - to change Auto Save Period (in epoch unit) (10000)
6 - to change the Delete Links Period (in epoch period) (1000000)
Any other key to return to the previous menu
```

Рисунок 3.8 - Параметри навчання нейронної мережі

```
Press:
1 - to train the network
2 - to process data
3 - to save the network
4 - to choose another network
5 - to change train settings
6 - to print neural network in console
Any other key - to exit

1
Enter the name of the file with a train data:
train
Enter the name of the file with a verification data:
verif
Epoch 1000; Error = 0.0106247915
Epoch 2000; Error = 0.010651475
Error is identical between 1000 epochs. Try to change network parameters.
```

Рисунок 3.9 - Результат навчання нейронної мережі

3.4 Тестування навченого нейромережевого додатка

Давайте розглянемо нейронну мережу на одній вибірці в тестовому наборі даних і на даних поза тестовою вибіркою (Рисунок 3.10).

```
Enter a number for the 1st Neuron: 4
Enter a number for the 2st Neuron: 1.150296
Enter a number for the 3st Neuron: 2.37962
Output # 1: 0.2412516
Press:
1 - to train the network
2 - to process data
3 - to save the network
4 - to choose another network
5 - to change train settings
6 - to print neural network in console
Any other key - to exit
1
Enter a number for the 1st Neuron: 1
Enter a number for the 2st Neuron: 0.9
Enter a number for the 3st Neuron: 0.8
Output # 1: 1.48999991
```

Рисунок 3.10 - Тестуування навчної нейромережі

Як видно з прикладів, нейронні мережі добре працюють з даними навчальних вибірок і довільних значень. Але майте на увазі, що нейронна мережа навчається на даних зі значеннями потоку води в діапазоні від 1 до 4. Для значень, менших або більших за вказане значення, нейронна мережа не навчена і видасть неправильні результати, чим більше числове відхилення, тим більша помилка.

3.5 Розв’язання поствленої в розділі 2 прикладної задачі

Ми будемо використовувати цю програму для створення нейронної мережі для вирішення наступних завдань. Наведено результати моделювання руху динамічної системи, що відхиляється від стану рівноваги, де декількарегуляторів мають різні параметри. Завдання регулювання – повернути систему в рівноважний стан. Кожна отримана траєкторія руху оцінюється певним значенням критерію якості. Тому, порівнюючи значення критеріїв якості, отримані при використанні кожного регулятора, можна вибрати той регулятор, який дає найкращий показник якості перехідного процесу при заданих початкових умовах руху.

Для цієї задачі ми будемо використовувати час кондиціонування як критерій якості - це інтервал часу від моменту, коли вхід отримує ступінчастий вплив (завдання, порушення) до моменту, коли значення кондиціонування відхиляється від встановленого значення, стає меншим, ніж деяке відносно невелике значення Digital delta (можна розглядати як нечутливу область контролера). Тому, особливо в задачі керування динамічними об’єктами, допустимо, щоб процес переходу завершився в якийсь момент, починаючи з якого відхилення заданого значення не перевищує 5% початкового значення відхилення (так звані 5% коридор). Тому для даної задачі в якості показника якості виберемо тривалість перехідного процесу до моменту відхилення системи від рівноважного стану не більше ніж на 5 % від початкового відхилення.

Для моделювання використовується динамічна система, яка описується наступними диференціальними рівняннями:

\

$$\ddot{x} = a * \dot{x} + b * x + c + U(x, \dot{x}),$$

де

x – регульована величина;

$\dot{x}, \ddot{x}$ – її перша та друга похідна за часом;

a, b, c – числові коефіцієнти;

$U(x, \dot{x})$ – закон керування (регулятор).

Програма моделювання генерує траєкторії та отримує стандартні значення якості для дискретного набору початкових відхилень і п’яти різних контролерів. Для кожної траєкторії розрахувати відповідне значення критерію якості.

Завдання, з яким стикаються додатки нейронної мережі, полягає в класифікації початкових зміщених точок фазової площини. Озброївшись навчальними даними щодо значень критерію якості, заданих кожним регулятором у дискретному наборі початкових відхилень, штучна нейронна мережа повинна визначити, який регулятор є оптимальним для будь-якої точки фазової площини.

Координати точки початкового відхилення на фазовій площині застосовуються до входу ШНМ, яка буде створена, а на виході повинні бути сигнали, які інтерпретуються як прогнозований рівень здійсненності з використанням кожного регулятора для даного відхилення. На базі цієї ШНМ необхідно створити додаток, який за результатами, отриманими від нейронної мережі, видасть кількість регуляторів, придатних для цього зміщення.

На рисунку 3.11 показано приклад використаних навчальних даних.

```
748|-10.00000|-10.00000$0
748|-10.00000|-9.59184$0
748|-10.00000|-9.18367$0
748|-10.00000|-8.77551$0
748|-10.00000|-8.36735$0
866|-10.00000|-7.95918$0
867|-10.00000|-7.55102$0
868|-10.00000|-7.14286$0
869|-10.00000|-6.73469$0
870|-10.00000|-6.32653$0
870|-10.00000|-5.91837$0
871|-10.00000|-5.51020$0
871|-10.00000|-5.10204$0
872|-10.00000|-4.69388$0
872|-10.00000|-4.28571$0
873|-10.00000|-3.87755$0
```

Рисунок 3.11 - Навчальна вибірка

На рис. 3.7 перше значення - індекс стандарту якості, друге і третє значення - координати точки на фазовій площині (діапазон значень від мінус десяти до десяти), а четверте значення - кількість коригувань для розрахунку індекс стандарту якості в даному точковому пристрої. У навчальних вибірках було використано 12500 значень.

Після перетворення навчальних вибірок у прийнятну для нейронної мережі форму ми отримуємо остаточні навчальні вибірки (рисунок 3.12).

```
-4.69388|1.42857$0|0|1|0|0
-10.00000|-5.91837$0|0|0|1|0
8.77551|-7.14286$0|1|0|0|0
-8.77551|-5.91837$0|0|0|1|0
1.02041|8.36735$0|1|0|0|0
5.10204|7.95918$1|0|0|0|0
-8.36735|-5.51020$0|0|0|1|0
-5.10204|-0.61224$0|0|0|0|1
6.73469|-9.18367$0|0|0|1|0
-6.73469|-7.95918$0|0|1|0|0
7.55102|-8.77551$0|0|0|1|0
9.18367|2.24490$0|1|0|0|0
-2.24490|-3.06122$0|0|1|0|0
3.06122|-2.24490$0|1|0|0|0
0.20408|2.65306$0|1|0|0|0
2.65306|0.61224$0|1|0|0|0
```

Рисунок 3.12 - Нормалізована навчальна вибірка

Тут перше і друге значення - це вхідні дані нейронної мережі (координати точки на фазовій площині), а останні п'ять значень - двійкове представлення виходу нейронної мережі, тобто регулятори, які потрібно використовувати, щоб зробити процес переходу найшвидшою кількістю. Оскільки при підготовці навчальних вибірок було обрано кондиціонер із найкращим значенням критерію якості, ми отримали 2500 різних даних у навчальних вибірках (важливо, що дані не в порядку).

Для цієї задачі ми припускаємо, що кількість прихованих шарів дорівнює 1, кількість нейронів дорівнює 50, а функція активації сигмовидної форми (приклад вказівки параметрів нейронної мережі показано на рисунку 3.13).

```
Enter the total number of layers, including input and output layers:
3
Enter the number of neurons in 1 layer:
2
Enter the number of neurons in 2 layer:
50
Choose an activation function for the 2 layer:
1 - PReLU
2 - ReLU
3 - Identity
4 - Sigmoid

4
Enter the number of neurons in 3 layer:
5
Choose an activation function for the 3 layer:
1 - PReLU
2 - ReLU
3 - Identity
4 - Sigmoid

4
```

Рисунок 3.13 - Параметри нейронної мережі

Щоб навчити нейронну мережу, ми будемо використовувати ті ж налаштування, що й у попередній нейронній мережі (див. рис. 3.8).

Виявилося, що час навчання цієї нейронної мережі був коротшим, ніж у попередньої нейронної мережі, оскільки, незважаючи на інший тип функції активації, кількість шарів і нейронів у цій нейронній мережі набагато менша. Результат нейронної мережі для першої точки в навчальній вибірці показано на рисунку 1. 3.14.

```
Enter a number for the 1st Neuron: -4.69388
Enter a number for the 2st Neuron: 1.42857
Output # 1: 1.8859595E-4
Output # 2: 0.73893666
Output # 3: 0.9966795
Output # 4: 2.422498E-9
Output # 5: 0.0011152417
```

Рисунок 3.14 - Результат обчислення параметрів регулятора нейромережею

З результатів видно, що регулятором, який найкраще підходить для даної точки на фазовій площині, є контролер номер 3.

Оскільки ця нейронна мережа має відповісти на питання, який регулятор слід використовувати, ми додамо нейронну мережу до модуля нейронної мережі, яка в результаті виведе номер регулятора.

```
public static void getResult() {
NeuralNetwork network = null; try {
network = getNeuralNetwork();
} catch (Exception e) {
System.out.println("Unhandled exception: " + e);
}
if(network != null) {
```

```
try (BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in))) {
System.out.println("Enter first value:"); FloatDataWrapper firstNumber = new
FloatDataWrapper(Float.valueOf(reader.readLine()));
System.out.println("Enter second value:"); FloatDataWrapper secondNumber =
new
FloatDataWrapper(Float.valueOf(reader.readLine()));
ArrayList<>();
network.processData();
ArrayList<FloatDataWrapper> inputData = new
inputData.add(firstNumber);                inputData.add(secondNumber);
network.setInputDataWrappers(inputData); ArrayList<FloatDataWrapper> results =
Integer bestResult = 0;
float maxNumber = results.get(0).getData();
for(int i = 1; i < results.size(); i++) { if(results.get(i).getData() > maxNumber)
{
maxNumber = results.get(i).getData(); bestResult = i;
}
}

System.out.println("Better to use regulator #" + (bestResult +

} catch (Exception e) {
System.out.println("Unhandled exception: " + e);
}
}
}
```

Результати роботи даного коду представлений на рис. 3.15.

```
Enter first value:
-4.69388
Enter second value:
1.42857
Better to use regulator #3
```

Рисунок 3.15 - Результат роботи нейромережевого додатка

У цьому прикладі координати точок вводяться користувачем вручну, але в реальній системі ці значення будуть передані в метод, а результати обчислень нейронної мережі будуть передані назад в систему для прийняття рішення. чи використовувати той чи інший регулятор.

Розділ 3 описує створення та роботу програм і методів навчання для генерації нейронних мереж заданої топології. Була створена нейронна мережа для визначення параметрів автоскалера, який використовується для керування проблемою водонапірної башти. Описує підготовку даних для визначеної нейронної мережі та навчання штучної нейронної мережі за допомогою зворотного поширення помилок.

Також розглядається розв'язок задачі вибору заданої множини авторегуляторів, яка ставиться як задача класифікації. Для вирішення цієї проблеми була створена та навчена друга штучна нейронна мережа з різною кількістю шарів і функціями активації для вирішення проблеми вибору регулятора для змодельованої одновимірної динамічної системи на основі метрики критерію якості.

Створена штучна нейронна мережа була перевірена на тестовому наборі даних. У результаті перевірки було виявлено, що обидві нейронні мережі забезпечують задовільні рішення своїх проблем.

ВИСНОВКИ

В результаті виконання магістерської роботи була створена програма для генерації нейронних мереж і нейромережевий додаток для визначення параметрів авторегулятора.

У першому розділі роботи досліджено актуальність проблеми використання нейронних мереж для керування авторегуляторами. Після проведених досліджень можна сказати, що використання нейронних мереж для визначення параметрів є одним із найперспективніших напрямів розвитку автоматичних регуляторів. Для такого регулятора достатньо отримати деякі експериментальні дані, які можна використовувати для навчання нейронної мережі.

Враховуючи складність і неточність визначення параметрівавторегулятора традиційними методами Циглера-Ніколса, формульними методами визначення налаштувань регулятора, настроювання регуляторів за номограмами та розрахунку налаштувань на основі частотних характеристик об’єкта (забезпечення запасу стабільності заданої системи)[2], на мою думку, використання нейронної мережі є найзручнішим і найпростішим рішенням.

У другому розділі дипломної роботи розглядаються задачі, які необхідно вирішувати за допомогою нейронних мереж. Для створення нейромережевого додатку було обрано задачу керування системою водопостачання на основітеорії автоматичного керування.

Для вирішення цього завдання вибирається топологія штучної нейронної мережі та її основні параметри, такі як: метод навчання – метод зворотного поширення помилки, тип функції активації – параметрична випрямлена лінійна функція, кількість прихованих шарів штучної нейронної мережі (три). а також кількість кожного нейрона (60, 40 і 20 відповідно), функцію суматора, кількість вхідних і вихідних нейронів мережі.

Розділ 3 дипломної роботи описує розробку програм для генерації нейронних мереж із заданою топологією та параметрами. Крім того, у третій

частині виконується підготовка даних і навчання нейронної мережі. За результатами тестування створеної нейронної мережі зроблено висновок, що застосування нейронної мережі має достатню точність для визначення параметрів авторегулятора. Однак є можливість підвищити точність за рахунок використання більшої кількості нейронів у прихованих шарах і урізноманітнення навчальних зразків.

Крім того, у третій частині за допомогою створеної програми генерується нейронна мережа для вирішення другої задачі – визначення кількості регуляторів із заданого набору регуляторів, які слід використовувати у разі певного зсуву в динаміці системи. повертає його до рівноваги і є найкращим показником якості перехідного процесу. Для цього генерується набір навчальних даних і перетворюється у формат, прийнятний для навчання ШНМ. На вхід цієї нейронної мережі надходять координати точки початкового відхилення на фазовій площині, на виході ми отримуємо сигнали, які інтерпретуються як відповідний ступінь для використання кожного регулятора для заданого відхилення. Створена ШНМ інтегрується в додаток, який публікує кількість регуляторів, які необхідно використовувати на основі обчислення координат нейронною мережею.

Програма, яка генерує нейронну мережу, також використовує метод видалення з’єднань і нейронів, які не використовуються для обчислення результату ШНМ.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017, 63 с.

2. Бондаренко С.Г., Сангінова О.В. Теорія автоматичного керування: метод. вказівки і завд. до викон. домашньої контр. роб. та самостійної роботи для студ. напр. підг. 6.050202 «Автоматизація та комп'ютерно-інтегровані технології», 2013, 102с.

3. ДСТУ 3008-95. Документація. Звіти у сфері науки і техніки. Структура і правила оформлення / Держстандарт України. – Вид. офіц. – [Чинний від 1995-02- 23]. – Київ, 2007. – 86с.

4. ГОСТ 2.301-68. Единая система конструкторской документации. Форматы. – Введ. 2002–01–01. – М. : Вид.-во стандартов, 2006. – 27 с.

5. Кузнецов М. И., «Основы электротехники» – 9-е издание, исправленное – Москва: Высшая школа, 1964, 560с.

6. Назаров А.В., Лоскутов А.И. Нейросетевые алгоритмы прогнозирования и оптимизации систем. Монография. СПб.: Наука и Техника, 2003, 384 с.

7. Поляков К.Ю. Теория автоматического управления для «Чайников», Санкт- Петербург, 2008, 80с.

8. Ясницкий Л.Н. Введение в искусственный интеллект. Учебное пособие для вузов. 2-е издание, испр. М., «Академия», 2008, 176 с.

9. James Keller, Derong Liu, David Fogel: Fundamentals of computational intelligence: neural networks, fuzzy systems, and evolutionary computation: John Wiley and Sons, 2016, 378 с.

10. Lionel Tarassenko, Mathematical background for neural computing, In Guide to Neural Computing Applications, Butterworth-Heinemann, New York, 1998, 285 с.

11. Anthony Martin, Artificial Neural Networks, 2003, 82 с.

12. Michael Nielsen. Neural Networks and Deep Learning, 2015.

13. Stegemann J. A., Buenfeld N. R., A Glossary of Basic Neural Network Terminology for Regression Problems. Neural Computing & Applications, 2006, 296 c.

14. Cybenko G.V., Approximation by Superpositions of a Sigmoidal function. У van Schuppen, Jan H. Mathematics of Control, Signals, and Systems. Springer International, 2006, 314 c.

15. Snyman Jan, Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms. Springer Science & Business Media, 2005.

16. Xu Bing, Wang Naiyan; Chen Tianqi, Li Mu, «Empirical Evaluation of Rectified Activations in Convolutional Network», 2015.

17. Gashler Michael S., Ashmore Stephen C., «Training Deep Fourier Neural Networks To Fit Time-Series Data», 2014.

18. Omidvar O.M., Elliot D.L. «Neural Systems for Controls», 1997, 357c.

19.Функции активации нейросети: сигмоида, линейная, ступенчатая, ReLu, than [Electronic resource]. – Access mode: https://neurohive.io/ru/osnovy- data-science/activation-functions/

20. Avinash Sharma V. «Understanding Activation Functions in Neural Networks» [Electronic resource]. – 12.09.2015. – Access mode: https://medium.com/the-theory-of- everything/understanding-activation-functions-in-neural-networks-9491262884e0

21. Fábio M. Soares, Alan M.F. «Souza Neural Network Programming with Java» [Electronic resource]. – 2016, 244 c. – Access mode: http://pzs.dstu.dp.ua/DataMining/bibl/practical/Neural%20Network%20Programming%2 0with%20Java.pdf

22. Maryna Hlaiboroda. «Нейрон, персептрон, кошка: фундаментальные отличия нейросетей.» [Electronic resource]. – 17.07.2018. – Access mode: https://blog.heyml.com/разновидности-нейронных-сетей-часть-1-12c4f7da8e32

23. Rashid T. «Make Your Own Neural Network», CreateSpace, [Electronic resource]. – 2016, 222 с – Access mode: https://www.twirpx.com/file/2089046/

24. Sagar Sharma. «Activation Functions in Neural Networks» [Electronic resource]. – 2017. Access mode: https://towardsdatascience.com/activation- functions-neural- networks-1cbd9f8d91d6

25. Панчук В.В., Богач І.В., Гуральник Ф.Б. Процеси та види тестування програмного забезпечення. Науково-технічна конференція факультету комп'ютерних систем і автоматики Вінницького національного технічного університету : матеріали L науково-технічної конференції ВНТУ, Вінниця, 2021. - URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2021/paper/view/12826.

26. Гуральник Ф.Б., Богач І.В., Панчук В.В. Автоматизація процесів регресійного тестування веб-додатків як підвищення швидкості і якості тестування. Науково-технічна конференція факультету комп'ютерних систем і автоматики Вінницького національного технічного університету : матеріали L науково-технічної конференції ВНТУ, Вінниця, 2021. - URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2021/paper/view/12827.

Додаток А

(обов'язковий)

ВНТУ

ЗАТВЕРДЖЕНО
Зав. кафедри КСУ ВНТУ,
д.т.н., доцент
_____ В'ячеслав КОВТУН
"__" _____ 2022 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи

Метод нейромережевого налаштування автоматичного регулятора комп'ютерно-інтегрованної системи

(тема)

Студент групи ЗАКІТ-21м

_____ Фредерік ГУРАЛЬНИК
Підпис *Ім'я ПРІЗВИЩЕ*

Керівник к.т.н., доцент каф. АІІТ

_____ Ілона БОГАЧ
Підпис *Ім'я ПРІЗВИЩЕ*

Консультант к.т.н., доцент, зав. каф. КСУ

_____ В' ячеслав КОВТУН
Підпис *Ім'я ПРІЗВИЩЕ*

Вінниця 2022

1. Назва та галузь застосування

1.1. Назва – Метод нейромережевого налаштування автоматичного регулятора комп’ютерно-інтегрованної системи

1.2. Галузь застосування – інофрмаційні технології.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ від “14” вересня 2022 року №203

3. Мета та призначення розробки.

Метою магістерської кваліфікаційної роботи є розрабка метода для автоматичного регулятора.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Бондаренко С.Г., Сангінова О.В. Теорія автоматичного керування: метод. вказівки і завд. до викон. домашньої контр. роб. та самостійної роботи для студ. напр. підг. 6.050202 «Автоматизація та комп’ютерно-інтегровані технології», 2013, 102с.

2. Vivek Bheda. Using Deep Convolutional Networks for Gesture Recognition in American Sign Language [Електронний ресурс] / Vivek Bheda, N. Dianna Radpour – Режим доступу до ресурсу: https://arxiv.org/ftp/arxiv/papers/1710/1710.06836.pdf.

3. Керівництво за технологіями об'єднаних мереж. 4 видання. - М .: Вільямс, 2005

4. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling [Електронний ресурс] / Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, Yoshua Bengio. – 2014. – Режим доступу до ресурсу: https://arxiv.org/abs/1412.3555

5. У.Одом "Офіційне керівництво Cisco по підготовці до сертіфікаціонним іспитів CCNA ICND2 200-101. Маршрутизація і комутація" (2016)

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- нейромережа для визначення параметрів для авторегулятора;
- налаштування регуляторів;
- формування результату.

5.2. Основні технічні вимоги до розробки:

- Windows 11;
- Matlab;
- Encog;
- InteliJ.

6. Стадії та етапи розробки.

6.1 Пояснювальна записка:

1. Аналіз методів, принципів, підходів і засобів реалізації задачі автоматизації процесами в об’єкті управління відповідно до теми кваліфікаційної роботи. Постановка задач дослідження «03» 09 2022 р.
2. Удосконалення технології прийняття рішень при автоматизації об’єкту управління «15 » 09 2022 р.
3. Визначення технічних характеристик системи «09» 10 2022 р.
4. Розробка програмного забезпечення системи «24» 10 2022 р.

6.2 Графічні матеріали:

1. Розробка UML – діаграми системи «04» 11 2022 р.
2. Розробка архітектури мережі «12» 11 2022 р.
3. Тестування програмного забезпечення «15» 11 2022 р.

7. Порядок контролю і приймання.

7.1. Хід виконання роботи контролюється керівником роботи. Рубіжний контроль провести до «28» 11 2022 р.

7.2. Атестація МКР здійснюється на попередньому захисті. Попередній захист магістерської кваліфікаційної роботи провести до «16» 12 2022 р.

7.3. Підсумкове рішення щодо оцінки якості виконання роботи приймається на засіданні ЕК. Захист магістерської кваліфікаційної роботи провести до «23» 12 2022 р.

Додаток Б
(обов'язковий)

ПРОТОКОЛ
ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ
НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Метод нейромережевого налаштування автоматичного регулятора комп'ютерно-інтегрованої системи»

Тип роботи: Магістерська кваліфікаційна робота
(БДР, МКР)

Підрозділ КСУ, ФІТА
(кафедра, факультет)

Показники звіту подібності Unicheck

Оригінальність 98,4% Схожість 1,6%

Аналіз звіту подібності (відмітити потрібне)

☐ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.

☐ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.

☐ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Галущак А.В.
(підпис) (прізвище, ініціали)

Ознайомлені з повним звітом подібності, який був згенерований системою Unichek щодо роботи.

Автор роботи _____ Гуральник Ф.Б.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Богач І.В.
(підпис) (прізвище, ініціали)

Додаток В

(довідковий)

Частковий лістинг вихідного коду програми

```
public interface ActivationFunction {


Float calculate(Float parameter);
Float getDerivative(Float parameter, Float result);
}


public class Identity implements ActivationFunction, Serializable {


public Float calculate(Float parameter) { return parameter;
}
public Float getDerivative(Float parameter, Float result) { return 1f;
}
}


public class PReLU implements ActivationFunction, Serializable {


public Float calculate(Float parameter) {
return (parameter < 0 ? 0.01f * parameter : parameter);

}
public Float getDerivative(Float parameter, Float result) { return (parameter < 0 ? 0.01f : 1);
}
}

public class ReLU implements ActivationFunction, Serializable {


public Float calculate(Float parameter) { return (parameter < 0 ? 0f : parameter);
}
public Float getDerivative(Float parameter, Float result) { return (parameter < 0 ? 0f : 1f);
}
```

```
}

public class Sigmoid implements ActivationFunction, Serializable {

public Float calculate(Float parameter) {
return (float) (1 / (1 + Math.exp(-parameter)));

}
public Float getDerivative(Float parameter, Float result) { return result * (1 - result);
}
}

public class NeuralNetwork implements Serializable {

private ArrayList<Layer> layers; private int[] numOfNeuronsInLayers;
private ArrayList<ActivationFunction> functions;

public NeuralNetwork(int[] numOfNeuronsInLayers, ArrayList<ActivationFunction> functions) {
this.numOfNeuronsInLayers = numOfNeuronsInLayers; this.functions = functions;

this.initLayers();
}

public void setInputDataWrappers(ArrayList<FloatDataWrapper> inputDataWrappers) {
ArrayList<Neuron> inputNeurons = layers.get(0).getNeurons(); for (int i = 0; i < inputNeurons.size(); i++) {
((InputNeuron) inputNeurons.get(i)).setInputData(inputDataWrappers.get(i));
}
}

public ArrayList<FloatDataWrapper> processData() { ArrayList<FloatDataWrapper> outputDataWrappers =
new ArrayList<>(); for (Neuron outputNeuron : layers.get(layers.size() - 1).getNeurons()) {
outputDataWrappers.add(new FloatDataWrapper(outputNeuron.getCalculatedOutput()));
}
return outputDataWrappers;

}
```

```
public void train(BufferedReader fileReader) throws IOException { String line;
while (true) {
line = fileReader.readLine();
if (line == null || line.isEmpty()) { break;
}
String[] parameters = line.split("\\$");

this.trainIteration(NeuralNetworkService.getDataWrappersFromString(parameters[0]),
NeuralNetworkService.getDataWrappersFromString(parameters[1]));
}

}


public float getError(BufferedReader fileReader) throws IOException {


ArrayList<FloatDataWrapper> actualResults = new ArrayList<>(); ArrayList<FloatDataWrapper>
expectedResults = new ArrayList<>(); String line;
while (true) {
line = fileReader.readLine();
if (line == null || line.isEmpty()) { break;
}
String[] parameters = line.split("\\$");


expectedResults.addAll(NeuralNetworkService.getDataWrappersFromString(parameters[ 1]));

actualResults.addAll(this.getResultsByParameters(NeuralNetworkService.getDataWrappe
rsFromString(parameters[0])));
}
float error = 0f;
for (int i = 0; i < actualResults.size(); i++) {
error += Math.pow(expectedResults.get(i).getData() - actualResults.get(i).getData(), 2);
}

return (error / expectedResults.size());
}


public int[] getNumOfNeuronsInLayers() { return numOfNeuronsInLayers;
}
public void deleteZeroLinksAndNeurons() { for(int i = layers.size() - 2; i > 0; i--) {
```

```
ArrayList<Neuron> neurons = layers.get(i).getNeurons(); for(int k = 0; k < neurons.size(); k++) {
for(int j = 0; j < neurons.get(k).getInputLinks().size(); j++) {
neurons.get(k).getInputLinks().removeIf(Link::isNeedToBeDeleted);
//pass the reference of the method and delete links with zero weight
}
neurons.removeIf(Neuron::isNeedToBeDeleted);  //delete if no input

links exist
}
}
}


private void initLayers() { this.layers = new ArrayList<>();
ArrayList<FloatDataWrapper> inputDataWrappers = new ArrayList<>(); for (int i = 0; i <
this.numOfNeuronsInLayers[0]; i++) {
inputDataWrappers.add(new FloatDataWrapper(0f));
}

layers.add(new Layer(inputDataWrappers, new Identity()));
//input layer




layers

for (int i = 1; i < this.numOfNeuronsInLayers.length - 1; i++) {  //hidden


layers.add(new Layer( this.numOfNeuronsInLayers[i], layers.get(layers.size() - 1).getNeurons(), false,
this.functions.get(i - 1)));

}
layers.add(new Layer(  //output layer this.numOfNeuronsInLayers[numOfNeuronsInLayers.length - 1],
layers.get(layers.size() - 1).getNeurons(),
true,
functions.get(functions.size() - 1)));
}
```

```
private void adjustPenalties(ArrayList<FloatDataWrapper> expectedResults, ArrayList<FloatDataWrapper>
actualResults) {

ArrayList<Neuron> outputNeurons = this.layers.get(this.layers.size() - 1).getNeurons();

for (int i = 0; i < outputNeurons.size(); i++) { float delta = expectedResults.get(i).getData() -
actualResults.get(i).getData();
outputNeurons.get(i).setPenalty(delta);

}
for (int i = layers.size() - 1; i > 0; i--) {

for (Neuron outputNeuron : this.layers.get(i).getNeurons()) { outputNeuron.adjustPenalties();
}
}
}


private void trainIteration(ArrayList<FloatDataWrapper> inputData, ArrayList<FloatDataWrapper>
outputData) {


this.adjustPenalties(outputData, this.getResultsByParameters(inputData));
}


private ArrayList<FloatDataWrapper> getResultsByParameters(ArrayList<FloatDataWrapper> inputData) {


this.setInputDataWrappers(inputData); return this.processData();
}


@Override
public String toString() {
StringBuilder stringBuilder = new StringBuilder(); stringBuilder.append("Neural network: ").append("{\n");
for(int i = 0; i < this.numOfNeuronsInLayers.length; i++) {
stringBuilder.append("\tLayer #").append(i + 1).append(" of
").append(this.numOfNeuronsInLayers.length).append(": ").append(this.layers.get(i));
}
stringBuilder.append("}\n"); return stringBuilder.toString();
}
```

```
}

public class Layer implements Serializable {

private ArrayList<Neuron> inputNeurons; private ArrayList<Neuron> neurons;
private ActivationFunction activationFunction;

public Layer(int numOfNeurons, ArrayList<Neuron> inputNeurons, Boolean isOutputLayer,
ActivationFunction activationFunction) {
this.inputNeurons = inputNeurons; this.activationFunction = activationFunction;
this.initNeurons(numOfNeurons, isOutputLayer);
}

public Layer(ArrayList<FloatDataWrapper> inputValues, ActivationFunction activationFunction) {
this.activationFunction = activationFunction; this.initInputNeurons(inputValues);
}

public void initInputNeurons(ArrayList<FloatDataWrapper> inputValues) { neurons = new ArrayList<>();
for(FloatDataWrapper singleValue : inputValues) { neurons.add(new InputNeuron(singleValue,
activationFunction));
}
}

public void initNeurons(int numOfNeurons, Boolean isOutputLayer) { neurons = new ArrayList<>();

for(int i = 0; i < numOfNeurons; i++) { neurons.add(isOutputLayer ? new OutputNeuron(inputNeurons,
activationFunction) : new Neuron(inputNeurons, activationFunction));
}

}

public ArrayList<Neuron> getNeurons() { return neurons;
}

@Override
```

```
public String toString() {
StringBuilder stringBuilder = new StringBuilder(); stringBuilder.append("\t{\n");
for(int i = 0; i < this.neurons.size(); i++) { stringBuilder.append("\t\tNeuron #").append(i + 1).append(" of
").append(this.neurons.size()).append(":     ").append(this.neurons.get(i));
}
stringBuilder.append("\t}\n"); return stringBuilder.toString();
}
}
```

```
public class NeuralNetworkSettings {

public static final float MIN_TRIGGERED_VALUE = 0.0000001f; //value on the output of the link that is equals to neuron's output multiplied be link's weight

public static float PENALTY_STEP = 0.00001f;        //parameter that determine how fast should weights be changed

public static int NUMBER_OF_EPOCH = 500000;     //one epoch is one pass through the file

public static float ALLOWABLE_ERROR = 0.0001f;              //value that determine whether network is accurate enough

public static int CHECK_ERROR_PERIOD = 1000; //number of epoch when verification should be performed

public static int AUTO_SAVE_PERIOD = 10000;      //number of epoch when neural network should be automatically saved to temporary file

public static int DELETE_LINKS_PERIOD = 1000; //number of epoch when links with zero outputs should be deleted
}
```

```
public class Neuron implements Serializable {

protected ArrayList<Link> inputLinks; protected ArrayList<Neuron> inputNeurons; protected float penalty;
protected float outputData; protected float inputData;
transient protected float sumOfWeights; protected ActivationFunction function;

public Neuron() {
this.inputNeurons = new ArrayList<>(); this.createLinks();
```

```
}

public Neuron(ArrayList<Neuron> inputNeurons, ActivationFunction function) { this.inputNeurons =
inputNeurons;
this.function = function; this.createLinks();
}

public float getCalculatedOutput() { this.initInputData(); this.setPenalty(0.0f);
this.outputData = function.calculate(this.inputData); return this.outputData;
}

public void createLinks() { this.inputLinks = new ArrayList<>();
for(Neuron singleInputNeuron : inputNeurons) { inputLinks.add(new   Link(singleInputNeuron));
}
}

public void adjustPenalties() {
float finalPenalty = this.penalty * function.getDerivative(this.inputData, this.outputData);
for(Link singleLink : inputLinks) { singleLink.adjustPenalty(finalPenalty);
}
}

public void setPenalty(float penalty) { this.penalty = penalty;
}

public float getPenalty() { return penalty;
}

protected void initInputData() { this.inputData = 0.0f; this.sumOfWeights = 0.0f; for(Link singleLink :
inputLinks) {
this.inputData += singleLink.getOutputData(); sumOfWeights += Math.abs(singleLink.getWeight());
}
}

public ArrayList<Link> getInputLinks() { return inputLinks;
```

```
}

public boolean isNeedToBeDeleted() { return inputLinks.isEmpty();
}

@Override
public String toString() {
StringBuilder stringBuilder = new StringBuilder(); stringBuilder.append("{\n\t\t\tactivation function: ").append(this.function.getClass().getName()).append("\n\t\t\tweights: ");

for(int i = 0; i < this.inputLinks.size(); i++) { stringBuilder.append(this.inputLinks.get(i).getWeight()).append("\t");
}
stringBuilder.append("\n\t\t}\n"); return stringBuilder.toString();
}
}

public class Link implements Serializable {

private float weight; private Neuron input;
transient private float outputData; transient private float neuronOutputData; transient private boolean isTriggered;
transient private boolean needToBeDeleted;

public Link(Neuron input) { this.isTriggered = false; this.needToBeDeleted = false;
this.weight = (float) (0.5 - Math.random() * 1); this.input = input;
}

public float getOutputData() {
this.neuronOutputData = input.getCalculatedOutput(); this.outputData = this.neuronOutputData * weight; this.isTriggered = Math.abs(this.outputData) >
NeuralNetworkSettings.MIN_TRIGGERED_VALUE; return outputData;

}

public void adjustPenalty(float penalty) {
```

```
        if (Float.isInfinite(this.outputData) || Float.isNaN(this.outputData)) { this.weight = (float) (0.5 - Math.random()
* 1);
        } else if (!isTriggered) { needToBeDeleted = true;
        } else {
        if(Float.isInfinite(penalty) || Float.isNaN(penalty)) { return;
        }
        needToBeDeleted = false; this.input.setPenalty(this.input.getPenalty() + penalty * weight); this.weight +=
NeuralNetworkSettings.PENALTY_STEP * penalty *
        this.neuronOutputData;
        }

        }


        public float getWeight() { return weight;
        }


        public boolean isNeedToBeDeleted() { return needToBeDeleted;
        }
        }
```

Додаток Г

(обов'язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

МЕТОД НЕЙРОМЕРЕЖЕВОГО НАЛАШТУВАННЯ АВТОМАТИЧНОГО РЕГУЛЯТОРА КОМП'ЮТЕРНО-ІНТЕГРОВАНОЇ СИСТЕМИ

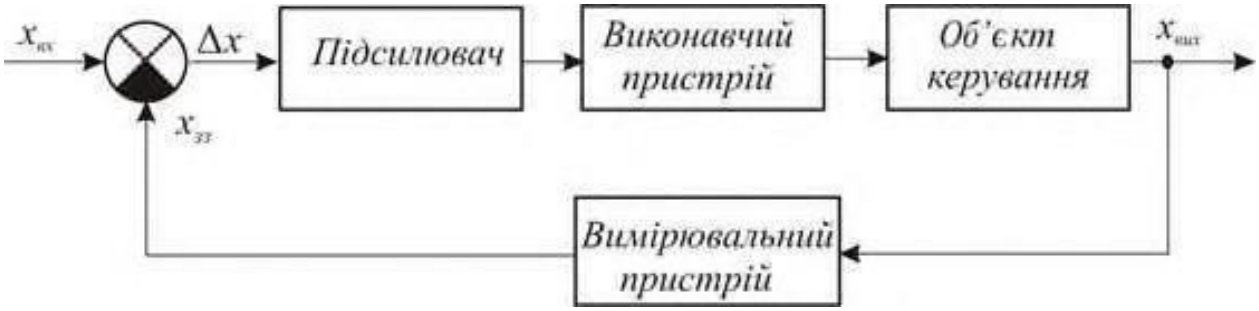
Студент групи №АКІТ-21м

Підпис Фредерік ГУРАЛЬНИК
Ім'я ПРІЗВИЩЕ

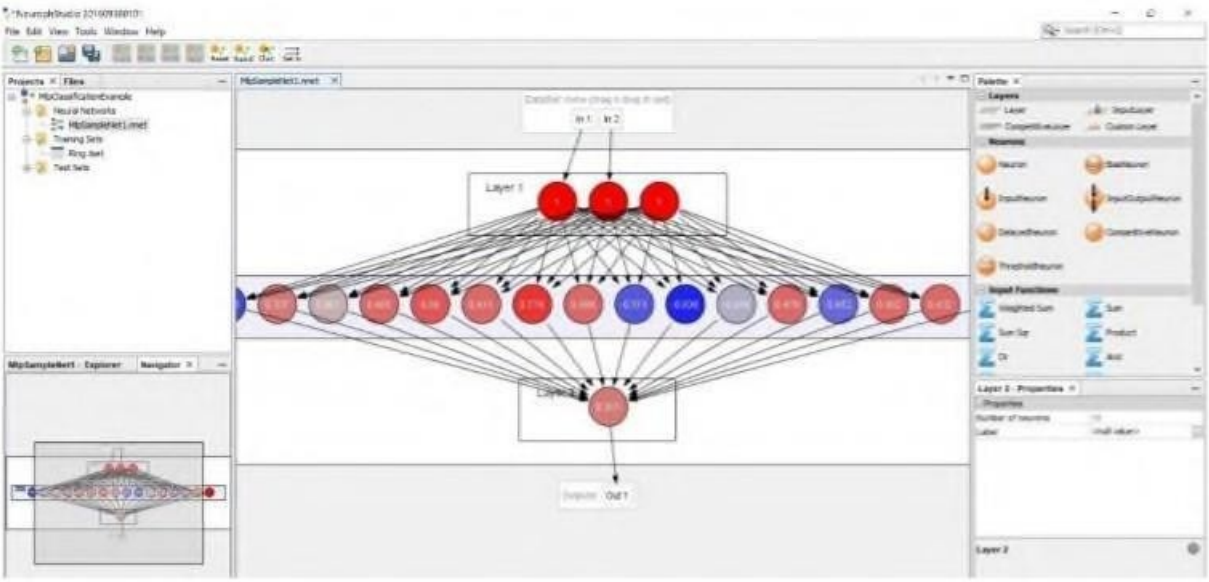
Керівник к.т.н., доцент каф.АІІТ

Підпис Ілона БОГАЧ
Ім'я ПРІЗВИЩЕ

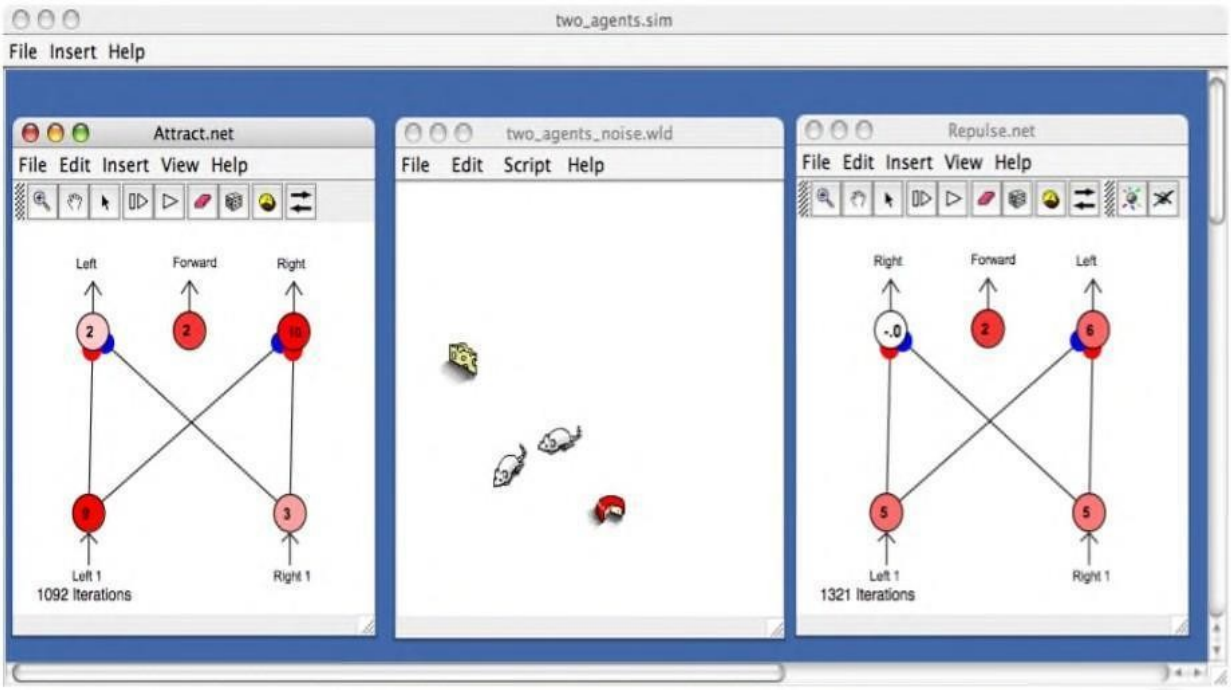
Структурна схема прямого автоматичного регулятора



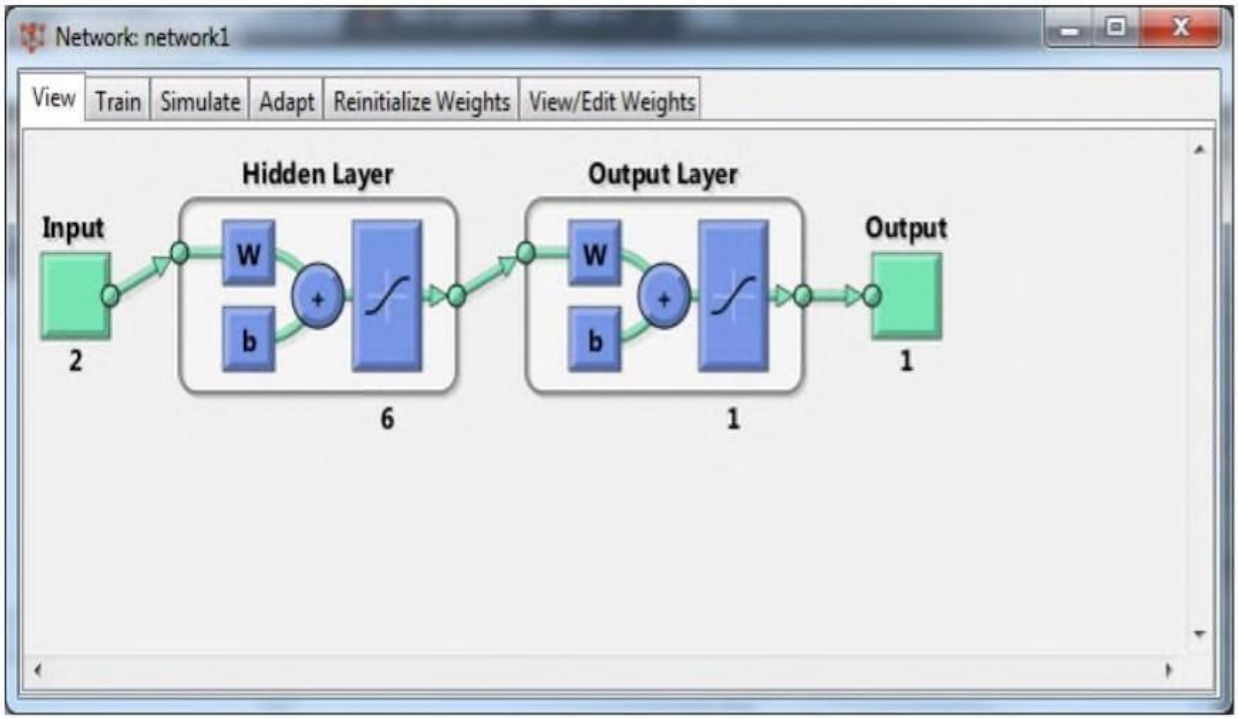
Структурна схема непрямого автоматичного регулятора



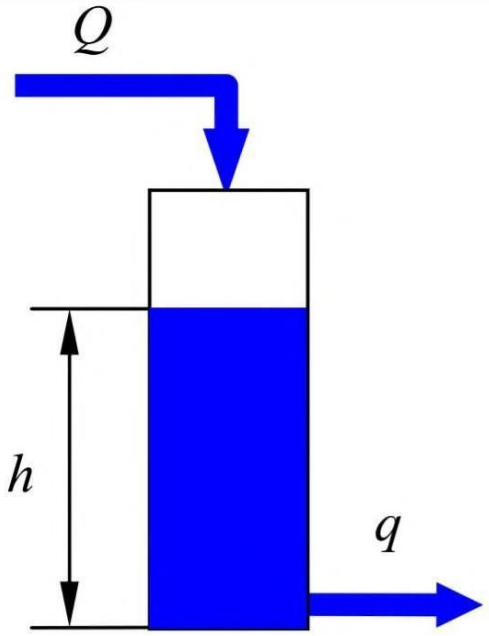
Програма Neuroph



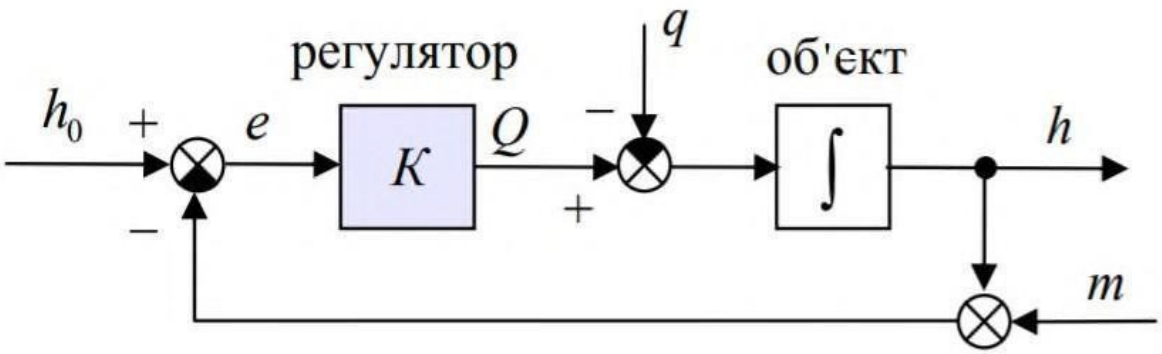
Програма Simbrain



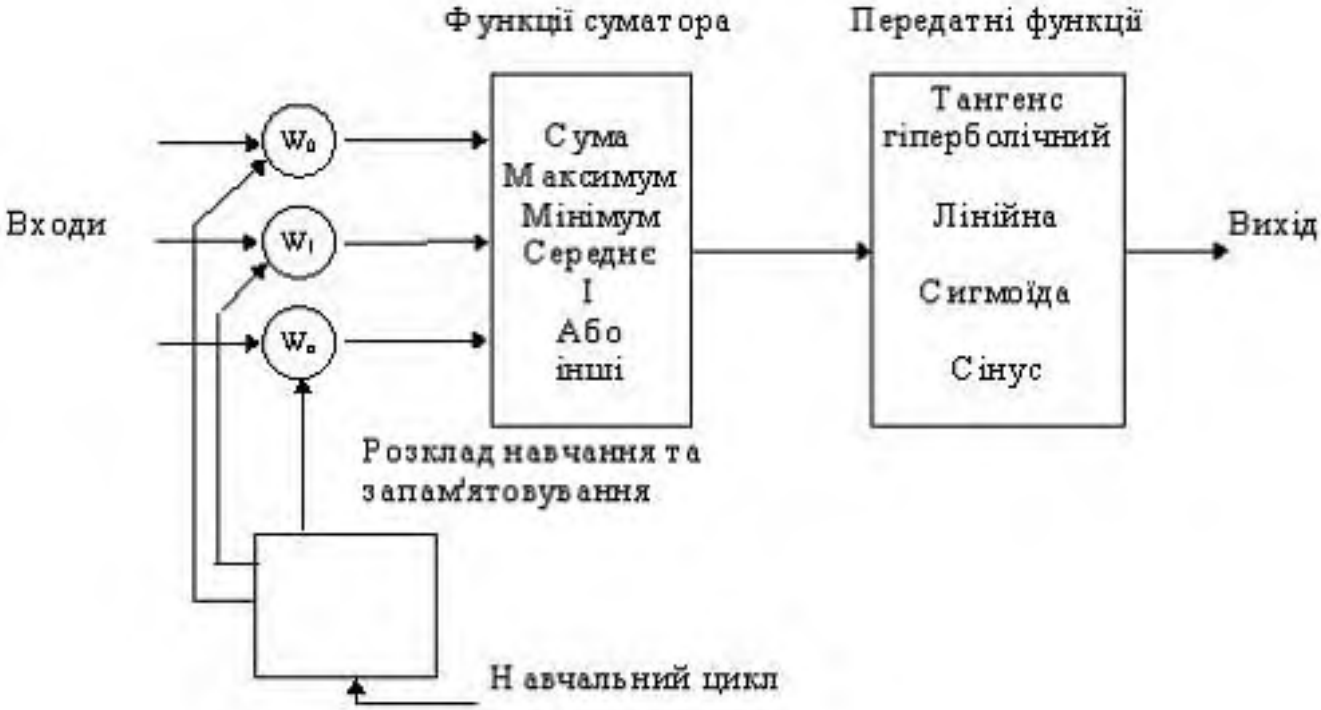
Модуль Deep Learning Toolbox в середовищі MATLAB



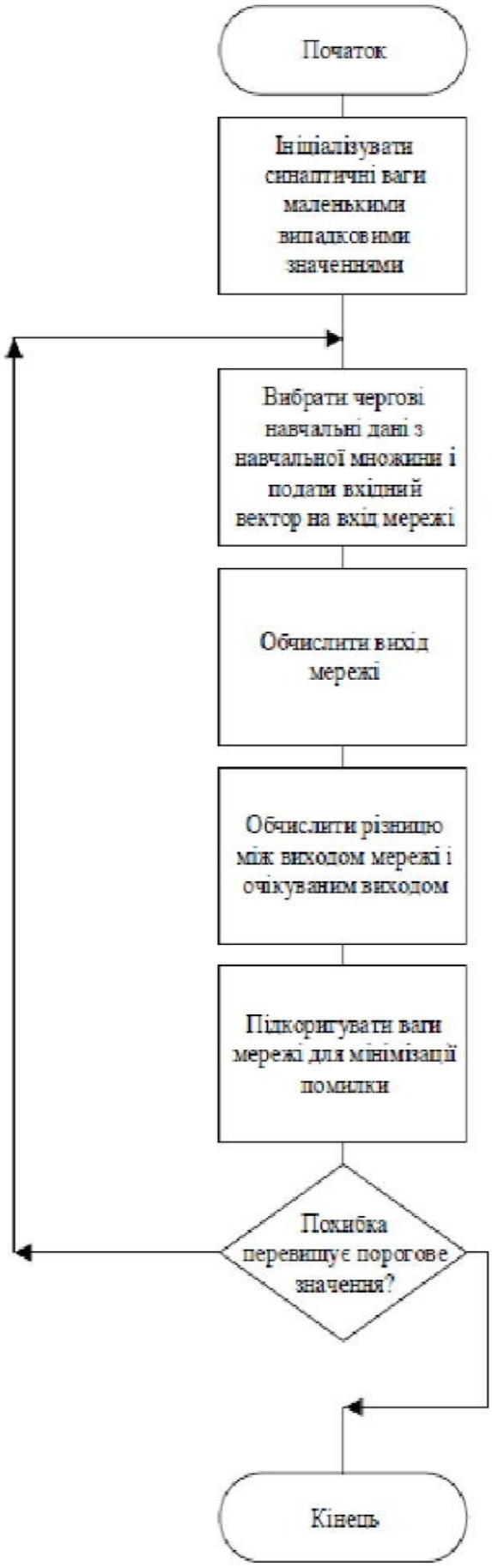
Графічна інтерпретація цистерни (люєкт управління)



Структурна схема системи управління



Модель елемента обробки



Алгоритм навчання нейромережі

```
Press:
1 - to train the network
2 - to process data
3 - to save the network
4 - to choose another network
5 - to change train settings
6 - to print neural network in console
Any other key - to exit

6
Neural network: {
    Layer #1 of 2:  {
        Neuron #1 of 2: {
            activation function: ActivationFunctions.Identity
            weights:
        }
        Neuron #2 of 2: {
            activation function: ActivationFunctions.Identity
            weights:
        }
    }
    Layer #2 of 2:  {
        Neuron #1 of 1: {
            activation function: ActivationFunctions.Sigmoid
            weights: 0.01380113 -0.24846941
        }
    }
}
```

Консоль з інформацією про нейромережу

```
Press:
1 - to change the Penalty Step (1.0E-11)
2 - to change the Check Error Period (in epoch unit) (10000)
3 - to change Number Of Epoch (1000000000)
4 - to change Allowable Error (1.0E-5)
5 - to change Auto Save Period (in epoch unit) (100000)
6 - to change the Delete Links Period (in epoch period) (500000)
Any other key to return to the previous menu
```

Опції управління навчанням нейромережі

```
1) Initialize existing Neural Network
2) Create new Neural Network
3) Exit
2
Enter the total number of layers, including input and output layers:
5
Enter the number of neurons in 1 layer:
3
Enter the number of neurons in 2 layer:
60
Choose an activation function for the 2 layer:
1 - PReLU
2 - ReLU
3 - Identity
4 - Sigmoid

1
Enter the number of neurons in 3 layer:
40
Choose an activation function for the 3 layer:
1 - PReLU
2 - ReLU
3 - Identity
4 - Sigmoid

1
Enter the number of neurons in 4 layer:
20
Choose an activation function for the 4 layer:
1 - PReLU
2 - ReLU
3 - Identity
4 - Sigmoid

1
Enter the number of neurons in 5 layer:
1
```

Встановлення параметрів нейронної мережі

```
Press:
1 - to train the network
2 - to process data
3 - to save the network
4 - to choose another network
5 - to change train settings
6 - to print neural network in console
Any other key - to exit

1
Enter the name of the file with a train data:
train
Enter the name of the file with a verification data:
verif
Epoch 1000; Error = 0.0106247915
Epoch 2000; Error = 0.010651475
Error is identical between 1000 epochs. Try to change network parameters.
```

Результат навчання нейронної мережі

```
Enter a number for the 1st Neuron: 4
Enter a number for the 2st Neuron: 1.150296
Enter a number for the 3st Neuron: 2.37962
Output # 1: 0.2412516
Press:
1 - to train the network
2 - to process data
3 - to save the network
4 - to choose another network
5 - to change train settings
6 - to print neural network in console
Any other key - to exit
1
Enter a number for the 1st Neuron: 1
Enter a number for the 2st Neuron: 0.9
Enter a number for the 3st Neuron: 0.8
Output # 1: 1.48999991
```

Тестування навчної нейромереж

```
Enter a number for the 1st Neuron: -4.69388
Enter a number for the 2st Neuron: 1.42857
Output # 1: 1.8859595E-4
Output # 2: 0.73893666
Output # 3: 0.9966795
Output # 4: 2.422498E-9
Output # 5: 0.0011152417
```

Результат обчислення параметрів регулятора нейромережею

```
Enter first value:
-4.69388
Enter second value:
1.42857
Better to use regulator #3
```

Результат роботи нейромережевого додатка