МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ
ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри
_____ Литвиненко О.Є.
«____» _____ 2020 р.

ДИПЛОМНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ «МАГІСТР»
ЗА СПЕЦІАЛЬНІСТЮ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: «Нейромережевий додаток визначення параметрів автоматичного регулятора»

Виконавець: студент СП-235М групи Покутній Дмитро Іванович

Керівник: доцент Глазок Олексій Михайлович

Нормоконтролер: Тупота Євгеній Вікторович

Київ 2020

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютеризованих систем управління

Спеціальність 123 «Комп'ютерна інженерія»

Освітньо-професійна програма «Системне програмування»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Литвиненко О.Є.

«____»_____ 2020 р.

ЗАВДАННЯ
на виконання дипломної роботи
Покутнього Дмитра Івановича

1. Тема роботи: «Нейромережевий додаток визначення параметрів автоматичного регулятора» затверджена наказом ректора від «27» серпня 2020 р. №1203/ст.

2. Термін виконання роботи: з 5 жовтня 2020 р. по 13 грудня 2020 р.

3. Вихідні дані до роботи: існуючі системи для проектування та навчання нейромереж, їх види та функціональні можливості. Різновиди функцій та способи їх апроксимації. Можливі варіанти комбінацій активаційних функцій нейронів. Засоби для написання та компіляції коду на мові програмування *Java*.

4. Зміст пояснювальної записки: аналіз автоматичних регуляторів, їх видів та параметрів. Аналіз існуючих способів визначення параметрів регуляторів. Побудова моделі нейромережі та розробка нейромережевого додатку. Навчання та тестування створеного додатку.

5. Перелік обов`язкового графічного матеріалу:

1. Схема алгоритму навчання нейромережі;
2. Схема алгоритму виконання обчислень нейромережею;
3. Параметри навчання нейромережі;
4. Результат навчання нейромережі;
5. Узагальнена діаграма класів програми.

6. Календарний план-графік

№ п/п	Етапи виконання дипломної роботи	Термін виконання етапів	Примітка
1	Ознайомитись з задачами автоматичного регулювання. Підготувати текст першого розділу пояснювальної записки.	5.10.20-16.10.20	
2	Проаналізувати існуючі способи визначення параметрів. Підготувати текст другого розділу пояснювальної записки	17.10.20-30.10.20	
3	Розробити та відладити програмний код додатку. Провести навчання і тестування створеного додатку. Підготувати текст третього розділу пояснювальної записки.	31.10.19-25.11.20	
4	Завершити оформлення пояснювальної записки. Оформити графічний матеріал.	26.11.19-08.12.20	
5	Пройти нормоконтроль. Підготувати доповідь та презентацію до захисту.	08.12.20-13.12.20	

7. Дата видачі завдання: «05» жовтня 2020 р.

Керівник дипломної роботи _____ Глазок О.М.

(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ Покутній Д.І.

(підпис випускника) (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Нейромережевий додаток визначення параметрів автоматичного регулятора»: 81 сторінка, 33 рисунки, 1 таблиця, 24 літературних джерела, 1 додаток.

ШТУЧНА НЕЙРОННА МЕРЕЖА, АВТОМАТИЧНИЙ РЕГУЛЯТОР, НЕЙРОМЕРЕЖЕВИЙ ДОДАТОК.

Об’єкт проектування – нейромережі.

Предмет проектування – нейромережевий додаток для визначення параметрів автоматичного регулятора.

Мета дипломної роботи – створення нейромережевого додатку для визначення параметрів автоматичного регулятора.

Метод проектування – розробка алгоритмів, написання програмного коду для створення додатку та його тестування.

Результати дипломної роботи рекомендується використовувати у модулях керування для визначення параметрів регулятора, а також для розробки нейромереж з різним набором шарів, нейронів та активаційних функцій.

ЗМІСТ

ВСТУП

Актуальність теми. В автоматичному регулюванні регулятор – це пристрій, який виконує функцію збереження визначеної характеристики. Він виконує діяльність з управління чи підтримки діапазону значень у системі.

Прикладами є регулятор напруги (це може бути трансформатор, коефіцієнт напруги трансформації якого можна регулювати, або електронна схема, що виробляє певну напругу), регулятор тиску, такий як дайвінг-регулятор, який підтримує свою потужність при фіксованому тиску нижче ніж його вхід, регулятор палива (який контролює подачу палива), тощо.

Універсального регулятора не існує, тому необхідно під кожну конкретну задачу вибирати регулятор та його параметри.

Для підбору параметрів використовуються різноманітні методики, які включають в себе експериментальні та математичні методи. Основними недоліками таких методів є їх відносна неточність та (або) складність. Окрім того, існує клас задач, де знаходження рішення вимагає значних обчислювальних витрат і збільшує час знаходження рішення, наприклад, в завданнях аеродинаміки при надзвукових швидкостях польоту літальних апаратів, коли виникають супутні завдання газової динаміки, теплопереносу, тощо.

Чисто теоретичне дослідження цих питань ускладнене через складність фізичних явищ і математичного апарату, що застосовується для їх опису. Експериментальне дослідження в натуральних умовах також представляє великі труднощі. Широко застосовується фізичне моделювання в лабораторних умовах, складність якого полягає в створенні адекватної моделі і необхідності використання дорогого устаткування. Ще більше вирішення подібних задач ускладняється впливом неврахованих параметрів, які можуть спотворювати кінцевий результат. Тому деякі задачі вирішуються базуючись на інтуїтивно-логічному мисленні, коли неможливо врахувати вплив багатьох факторів через високу складність об’єкту керування, але такий метод прогнозування дає невисоку точність, а інколи зовсім протилежні результати.

Для вирішення проблеми визначення параметрів автоматичного регулятора можуть застосовуватися спеціалізовані системи, які реалізують нейромережеві технології. Важливою відмінністю даних систем є те, що характер обробки інформації залежить не від визначених алгоритмів а від розподілу зв'язків між нейронами, тому така система може підлаштовуватися під нові дані і доволі швидко пристосовуватися до нових вихідних даних. Окрім того, стан кожного окремого нейрона визначається впливом станів багатьох інших нейронів, пов'язаних з ним, тому втрата одного або декількох зв'язків не робить істотного впливу на результат роботи системи в цілому, завдяки чому забезпечується висока надійність системи.

На практиці, при розробці подібних нейромереж проектується певна модель об'єкту з визначеними вхідними і вихідними параметрами і після навчання нейромережі, вона може видавати результат з доволі високою точністю. Ця точність залежить від правильності підбору вхідних і вихідних параметрів, вибору топології мережі, кількості шарів нейронів і т.д. Проте, при правильному підборі вищезазначених параметрів нейромережі, ця нейромережа може прогнозувати параметри не тільки для однієї, заздалегідь визначеної задачі, а для цілого ряду подібних задач.

Мета і завдання виконання дипломної роботи. Мета дипломної роботи – створити програму, що може бути використана для генерації нейромережевого додатку для визначення параметрів автоматичного регулятора.

Для виконання поставленого завдання необхідно:

- ознайомитись з проблемою визначення параметрів автоматичного регулятора за допомогою аналітичних, математичних та експериментальних методів;
- ознайомитись з існуючими рішеннями для створення нейромереж;
- визначитися з конкретною прикладною задачею, яку необхідно вирішити за допомогою нейромережі;
- визначити топологію та основні параметри нейромережі;
- реалізувати програму для створення нейромереж за визначеними топологією та параметрами;

– згенерувати за допомогою створеної програми нейромережу, що буде виконувати визначення параметрів автоматичного регулятора;
– навчити ШНМ та проаналізувати результат роботи нейромережі.

Об’єкт і предмет дослідження. Об’єктом дослідження даної дипломної роботи є нейронні мережі. Предмет розробки та дослідження – нейромережевий додаток для визначення параметрів автоматичного регулятора.

Методи дослідження. Спосіб визначення параметрів автоматичного регулятора досліджується за допомогою математичних та аналітичних методів; запропоновано топологію та параметри нейромережі для вирішення поставленого завдання; програмно реалізовано багатошаровий персептрон з використанням методу зворотного поширення похибки для навчання ШНМ, при цьому використовувалися проектування, написання, відладка коду, тестування програмного продукту.

Практичне значення отриманих результатів. Розроблено програму для генерації та навчання нейромереж з топологією «багатошаровий персептрон» і зворотнім методом поширення похибки. Дана програма дозволяє створити ШНМ з будь-якою кількістю прошарків на нейронів. Активаційна функція може бути вибрана окремо для кожного шару з чотирьох доступних, що дозволяє згенерувати різні комбінації нейромереж. Дану програму використано для створення нейромережі для визначення параметрів автоматичного регулятора водопостачання та нейромережу для визначення номера регулятора, який доцільно використати в системі керування динамічною системою при заданому відхиленні.

Особистий внесок випускника. Всі результати, представлені у дипломній роботі, отримані випускником особисто. Програма в цілому, її компоненти, класи та методи являються власною розробкою автора диплома. Окрім того, навчальний та тестовий набори даних, підготовлені та приведені до відповідного формату, а також нейромережі визначеної топології та параметрів, створені автором в межах даної дипломної роботи.

Прогнозні припущення про розвиток об’єкту та предмету дослідження. Створену програму можна використовувати для генерації ШНМ, їх навчання та подальшого використання в якості модуля в інших програмах та додатках. Одну з

створених нейромереж можна використовувати для визначення параметрів автоматичного пропорційного регулятора з областю значень від нуля до чотирьох або, при додатковому навчанні на відповідних навчальних даних, для визначення параметрів автоматичного пропорційного регулятора з будь якою додатною областю значень. Другу ШНМ можна використовувати для визначення найкращого регулятора з заданого набору, який може забезпечити найшвидше повернення динамічної системи до початкового відхилення.

РОЗДІЛ 1
АВТОМАТИЧНІ РЕГУЛЯТОРИ

1.1. Види автоматичних регуляторів

Система автоматичного регулювання (САР) – це така система автоматичного керування, задача якої полягає у підтримці на заданому рівні або зміні по заданому закону вихідної величини *Y(t)* об'єкта.

Система автоматичного регулювання складається з регульованого об'єкта та елементів управління, які впливають на об'єкт при зміні однієї або декількох регульованих змінних. Під впливом вхідних сигналів (управління або обурення), змінюються регульовані змінні. Мета ж регулювання полягає у формуванні таких законів, при яких вихідні регульовані змінні мало відрізнялися б від необхідних значень. Рішення даного завдання в багатьох випадках ускладнюється наявністю випадкових збурень (перешкод). При цьому необхідно вибирати такий закон регулювання, при якому сигнали керування проходили б через систему з малими спотвореннями, а сигнали шуму практично не пропускалися.

Автоматичне регулювання забезпечує оптимальну продуктивність динамічних систем, надзвичайно підвищує продуктивність праці та позбавляє необхідності виконувати одне і те ж завдання знову і знову.

Автоматичний регулятор – це пристрій, компонент у системі автоматичного регулювання, що виробляє керуючий сигнал для зміни (регулювання) вихідного параметра.

До системи автоматичного регулювання також входить регульований об’єкт – система, для роботи якої на заданому режимі потрібно застосовувати регулюючі дії.

Об'єктом регулювання може бути будь-яка динамічна система або її модель. Стан об'єкта характеризується деякими кількісними величинами, що змінюються в часі, тобто змінними стану. У природних процесах в ролі таких змінних може виступати температура, щільність певного речовини в організмі, курс цінних паперів і т. Д. Для технічних об'єктів це механічні переміщення (кутові або лінійні) і їх

швидкість, електричні змінні, температури і т. д. Аналіз і синтез систем управління відбувається за допомогою засобів спеціального розділу математики - теорії управління.

Автоматичний регулятор виконує завдання, яке визначається системою керування. Регулятор визначає регулюючу дію, яка через виконавчий елемент і регулюючий орган діє на об’єкт регулювання. При роботі системи регулювання чутливий механізм постійно виконує вимірювання регульованої величини, а регулятор може діяти на виконавчий механізм постійно або мати перервний (дискретний) характер дій. В залежності від характеру регулюючої дії на об’єкт регулювання або залежно від конструкції регулятора вони поділяються на регулятори прямої (рис. 1.1) і непрямої (рис. 1.2) дії [2].

Рис. 1.1. Структурна схема прямого автоматичного регулювання

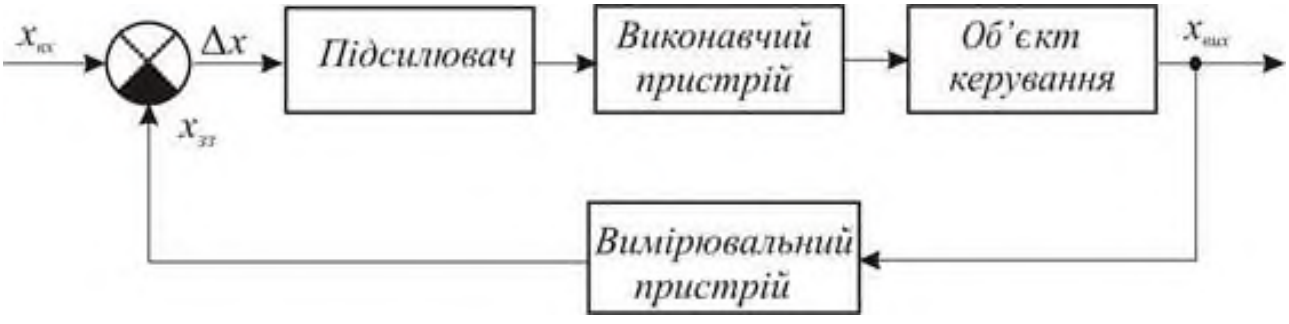

Рис. 1.2. Структурна схема непрямого автоматичного регулювання

До регуляторів прямої дії відносять такі регулятори, у яких зусилля, необхідне для переміщення регулюючого органу, виникає за рахунок зміни вихідного параметра без підведення додаткової енергії. В цьому випадку датчик (чутливий елемент) є одночасно і виконавчим механізмом.

При прямому регулюванні автоматичні регулятори мають одну ступінь свободи. Складається з одного елементу, який реагує на зміну регульованої величини і безпосередньої дії на регулюючий орган об’єкту, який регулюється.

На практиці ширшого застосування набули регулятори непрямої дії. Це регулятори, у яких зусилля, необхідне для переміщення регулюючого органу, виникає за рахунок впливу окремого виконавчого механізму. Ці регулятори класифікуються за видом джерела енергії, що використовується для переміщення виконавчого механізму на:

– механічні;
– пневматичні;
– гідравлічні;
– електричні;
– комбіновані;
– змішані, наприклад пнемо-гідравлічні.

В залежності від числа регульованих режимів можна виділити:

– однорежимні;
– дворежимні;
– всережимні;
– універсальні.

Крім того, регулятори класифікуються на релейні, безперервні та імпульсні. Релейні регулятори називають ще позиційними.

Регулятори поділяють також на екстремальні і стабілізаційні. Екстремальні регулятори можуть використовуватися на об'єктах, що характеризуються екстремальною статичною характеристикою, вигляд якої може залежати від зовнішніх впливів і змінюватися у часі.

Найбільше поширення отримали стабілізуючі регулятори, тобто регулятори, які виконують корегування роботи об’єкта регулювання, забезпечуючи відхилення не більше, ніж задане.

Для постійно модульованого управління контролер зворотного зв'язку використовується для автоматичного управління процесом або роботою. Система

управління порівнює значення або стан технологічної змінної, що контролюється, з бажаним значенням або заданим значенням, і застосовує різницю як контрольний сигнал, щоб привести вихідну змінну технологічного процесу установки до того ж значення, що і задане значення.

Залежність керуючого сигналу, що виробляється регулятором, від сигналу розбалансу у часі, визначається законом регулювання, серед яких найширшого застосування набули:

– пропорційний закон регулювання;
– інтегральний закон регулювання;
– пропорційно-інтегральний закон регулювання;
– пропорційно-диференціальний закон регулювання;
– пропорційно-інтегрально-диференціальний закон регулювання.

Відповідно, за законом регулювання, стабілізуючі регулятори класифікуються на інтегральні (І), пропорційні (П), пропорційно-інтегральні (ПІ), пропорційно-диференціальні (ПД) і пропорційно-інтегрально-диференціальні (ПІД).

Закон регулювання формується за допомогою зворотних зв'язків. З урахуванням динамічних властивостей об'єкта керування він визначає вид і якість перехідного процесу в САР.

Регулятори із зворотним зв’язком іноді називаються регуляторами із жорстким зворотним зв’язком. Крім цього, є ще регулятори з комбінованим зворотним зв’язком. Комбінований зворотний зв’язок ще в літературі називають ізодромним .

Об’єкт, що регулюється (регульований об’єкт), незалежно від його типу (теплова машина, тепловий двигун, парова машина), обладнаний автоматичним регулятором (системою автоматичного регулювання), має свої статичні і динамічні характеристики. Ці характеристики можна описати відповідними залежностями.

В залежності від типу статичної (регуляторної характеристики) САР поділяють на статичну і астатичну.

До статичних систем регулювання відносяться системи, які підтримують значення регульованого параметру в деяких чітко визначених межах Δh або Δn при нерівномірності регулятора δ.

Астатичні системи регулювання підтримують тільки одне постійне значення $n_{ном}$ регульованого параметру. Наприклад, астатичні характеристики дизель-електричних станцій забезпечують постійну частоту струму 50 *Гц* при незмінній частоті обертання колінчатого вала двигуна ($n=const$). САР, яка характеризується однією регульованою величиною одного регулятора, називається одноконтурною системою (одноімпульсною).

Автоматичні регулятори, які реагують не тільки на відхилення регульованого параметру n (оберти), а і на його похідну ω – прискорення, називають двоімпульсними: n, ω.

Є системи, у яких одночасно регулюються декілька параметрів, наприклад у парових турбінах контролюються і регулюються: обертання ротора турбіни, тиск пару, температура пару на вході та на виході та інші електростатичні параметри.

1.2. Математичний і графічний опис процесу автоматичного регулювання

Математично процес регулювання характеризується зміною регульованої величини x_1 в часі t. Процес регулювання $x_1(t)$ можна показати у вигляді графіка, який також називають кривою процесу регулювання. Приклад такого графіка наведено на рис. 1.3

При відсутності зовнішнього і внутрішнього збудження на систему автоматичного регулювання в ідеальному випадку ми отримуємо пряму $x_1^0(t)$, яку показано на рис. 1.3 пунктирною лінією. В реальних умовах експлуатації на систему постійно діють збудження. При цьому крива процесу регулювання $x_1(t)$, яка відображає фактичне значення регульованої величини, має бути близько до прямої $x_1^0(t)$.

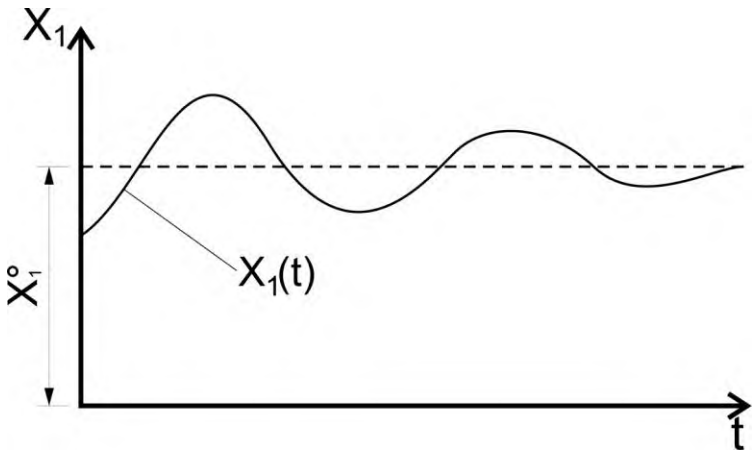

Рис. 1.3. Графік процесу регулювання

Це означає, що які б збудження на систему не діяли, автоматичний регулятор повинен весь час тримати регульований параметр (величину) біля заданого значення x_1^0. Крива процесу регулювання показує на скільки добре дана система справляється зі своєю задачею. В технічних вимогах до САР вказується, за які межі не може виходити відхилення фактичного значення регульованої величини, тобто крива $x_1(t)$ від встановленого значення прямої $x_1^0(t)$. Наприклад на 2% від величини x_1^0.

1.3. Параметри автоматичного регулятора

До параметрів регулятора та його ланок відносяться:

– нерівномірність регулятора δ;
– коефіцієнт підсилення;
– передаточне число;
– час запізнювання щодо передачі сигналу;
– момент інерції;
– індуктивність;
– ємність;
– опір електричної ланки та інше.

Відповідність вищевказаних параметрів регулятора заданим ще не гарантує придатності регулятора до виконання своїх функцій. При невдалому виборі параметрів регулятор може бути таким, що САР не заспокоює систему, а навпаки – її

розгойдує. При цьому крива процесу регулювання буде відходити від заданої величини (рис. 1.4, 1.5).

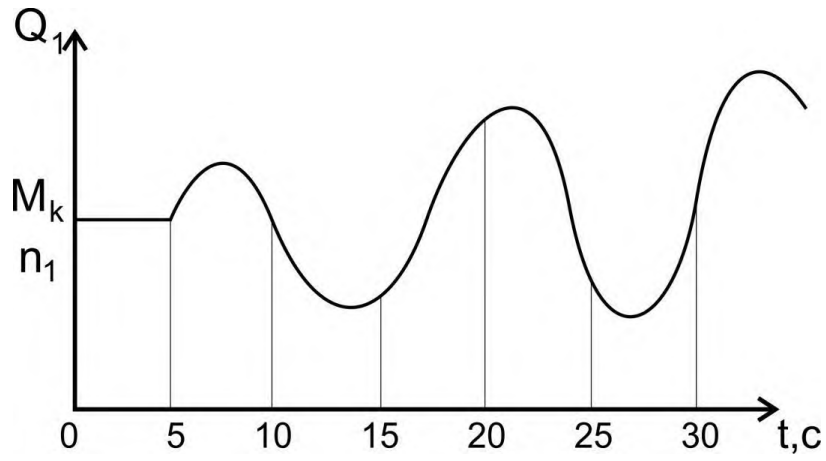

Рис. 1.4. Нестійка регульована система, траєкторія руху якої розходиться

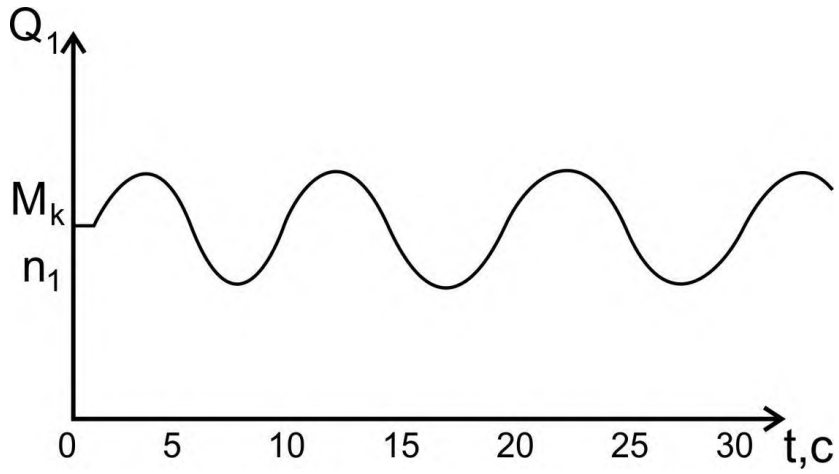

Рис. 1.5. Нестійка регульована система з постійною амплітудою коливань

Розраховувати і перевіряти експериментально потрібно не тільки у врівноваженому режимі роботи САР машини, а також перехідні процеси та інші динамічні режими, коли є змінні збурення на об’єкт регулювання, а через нього і на САР. Серед різних видів перехідних процесів необхідно знати такі:

– при включенні машини і САР;
– при переналагодженні системи на нові значення регульованої величини;
– при збуджуючій дії у вигляді скачкоподібної зміни навантаження системи;
– при виникненні збуджуючої дії або заданого різноманітного навантаження;

– в момент включення САР може бути велике початкове відхилення поточного значення регульованої величини (параметра) від того значення, яке необхідно встановити в результаті роботи регулятора.

Для швидкої ліквідації великого начального непогодження двох параметрів необхідно швидке регулювання регулятора на початку процесу. Таке відхилення регульованої величини у протилежну сторону називається закидом (перерегулювання). Технічна вимога до САР передбачає швидке затухання перехідного процесу, який впливає на якість роботи системи, що регулюється.

1.4. Визначення нейромережі та нейрона

Нейронна мережа (штучна нейронна мережа) – це спосіб використання фізичного обладнання або комп’ютерного програмного забезпечення для моделювання обчислювальних властивостей, аналогічних тим, які були постульовані для реальних мереж нервів, таких як здатність вчитися та зберігати зв’язки. Нейронна мережа може плавно апроксимувати та інтерполювати дані з багатьох змінних, які в іншому випадку можуть вимагати величезних баз даних, у компактний спосіб.

Нейронна мережа має велику кількість процесорів (прошарків). Ці процесори працюють паралельно, але розташовані у вигляді рівнів. Перший рівень отримує необроблені дані, аналогічні тому, як зоровий нерв отримує необроблену інформацію у людей. Кожен наступний рівень потім отримує вхідні дані від рівня перед ним і потім передає свої вихідні дані рівню після нього. Останній рівень обробляє остаточний результат.

Маленькі вузли складають кожен рівень. Вузли тісно пов'язані з вузлами на рівні до і після. Кожен вузол в нейронній мережі має свою власну область знань, включаючи правила, які він запрограмував, і правила, які він вивчив сам. Ключовим фактором ефективності нейронних мереж є те, що вони надзвичайно пристосовані та навчаються дуже швидко. Кожен вузол оцінює важливість вхідних даних, які він отримує від вузлів до нього. Входи, які вносять найбільший вклад в правильний вихід, отримують найбільшу вагу.

Популярність нейромереж у різних сферах останнім часом зумовлена, зокрема:

– даними – величезна кількість доступних даних, зібраних за останнє десятиліття, багато в чому сприяла популярності глибокого навчання. Це дозволило ШНМ по-справжньому показати свій потенціал, оскільки вони визначають результат з меншою похибкою при збільшенні кількості даних;
– комп'ютерною потужністю – обчислювальна потужність, яка стала доступною в останні десятиліття, дозволяє обробляти більше даних з більшою швидкістю, що призводить до покращення і пришвидшення навчання ШНМ;
– алгоритмами – успіхи, досягнуті в області розробки алгоритмів, дозволяють вибрати для конкретної задачі відповідну топологію нейромережі, а розроблені методи навчання дозволяють швидше навчати нейромережу;
– маркетингом – незважаючи на те, що нейронні мережі існують ще з 1944 року, справжню популярність вони отримали після створення декількох успішних продуктів, наприклад роботу *Sophia* від компанії *Hanson robotics*, створення якого призвело до збільшення висвітлення даної теми в засобах масової інформації.

ШНМ складається з нейронів і зв'язків між ними. Різні топології нейромереж використовують різну кількість нейронів та прошарків нейронів, а також різні зв'язки між ними (наприклад, у повністю рекурентній ШНМ кожен нейрон мережі з'єднаний з кожним іншим нейроном мережі, а у персептроні – кожен нейрон прошарку з'єднаний з кожним нейроном попереднього прошарку).

Нейрони – це складова ШНМ, яка обчислює задану функцію суматора, виконуючи операцію над вхідними даними і повертаючи результат як вихід нейрона. Нейрони одного рівня поєднуються у прошарок.

Зв'язки між нейронами мають певну вагу – число, на яке потрібно помножити результат обчислення нейрона попереднього прошарку. Результат множення далі передається на вхід нейрона наступного шару. При цьому, в більшості випадків, вага зв'язків – це єдиний параметр, який змінюється під час навчання ШНМ. Інші

параметри мережі, такі як функція суматора і активаційна функція нейрона залишаються незмінними.

1.5. Використання ШНМ для визначення параметрів регулятора

Найперспективнішими напрямами розвитку систем керування складними електротехнічними системами сьогодні, поряд з системами нечіткої логіки, є нейронні мережі. Ці напрями розвитку систем керування належать до категорії інтелектуальних систем керування, які дають змогу отримати хороші результати в разі неповної математичної моделі об’єкта керування або у випадку, коли параметри об’єкта керування змінюються в процесі функціонування.

Останнім часом з’являється все більше праць з питань реалізації систем автоматичного регулювання за допомогою нейромереж. Однак здебільшого в них розглядають системи, які так чи інакше повторюють класичний підхід до реалізації регулятора на підставі класичних законів регулювання, у яких керувальний вплив пропорційний до відхилення, інтеграла чи похідної відхилення регульованої величини від її заданого значення. Водночас досить перспективним вважається інший спосіб побудови регулятора – з прогнозуванням значення регульованого параметра, якого він набув би через певний час після можливого заданого впливу на регульовану систему.

Рішення про вплив приймають на підставі порівняння прогнозованого і бажаного значення параметра регулювання. Розв’язування задачі прогнозування покладають на нейромережу, попередньо налаштовану на орієнтовні параметри системи з можливістю адаптивного їх уточнення.

На даний момент існує доволі багато розроблених мереж для керування певною системою. Проте, дані ШНМ здебільшого знаходяться в закритому доступі і розраховані для використання у конкретній системі. Для кожної задачі необхідно розробляти власну нейромережу, оскільки параметри системи та регуляторів можуть кардинально відрізнятися від системи до системи.

Для розробки ШНМ використовують різні програмні системи, які здебільшого мають в своєму складі багато варіацій топологій і параметрів ШНМ, проте, створена за допомогою такої програмної системи ШНМ, не завжди може бути використана поза програмною системою, як окремий модуль.

1.6. Огляд існуючих рішень

В мережі Інтернет можна знайти достатньо теорії щодо створення нейромереж для регулювання процесом або системою, проте готові рішення знайти дуже важко. Більшість нейромереж, призначених для керування або визначення параметрів автоматичних регуляторів розробляються для конкретних задач і знаходяться у закритому доступі.

Для створення ШНМ використовуються різноманітні програмні системи, проте всі вони мають певні обмеження – неможливість створити окремий нейромережевий модуль, мають у своєму складі надлишкові типології нейромереж, не дають можливості запровадити власну сумуючу функцію, мають обмежений набір активаційних функцій, тощо. Розглянемо деякі з таких систем.

Neuroph – система для нейромережевого розпізнавання, написана на мові *Java* (рис. 1.6). Складається з *Java API*, що включає в себе основні класи, утиліти і реалізацію конкретних типів нейронних мереж. Також включає в себе середовище *Neuroph Studio*, реалізовану на платформі *NetBeans*.

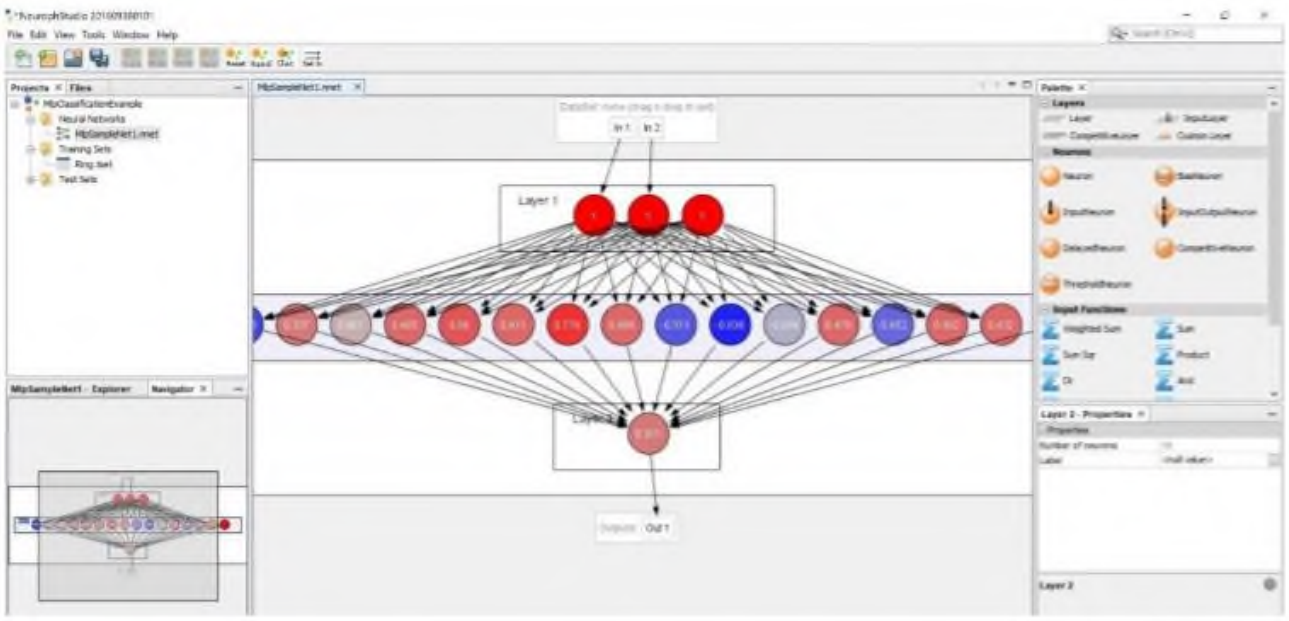

Рис. 1.6. Інтерфейс програми *Neuroph*

У середовищі реалізовані наступні нейромережеві архітектури:

– адалайн;
– персептрон;
– багатошаровий персептрон з алгоритмом зворотного поширення помилки, моментом;
– мережа Хопфілда;
– двостороння асоціативна пам'ять;
– мережа Кохонена;
– мережа Хебба;
– *RBF* нейронна мережа.

У систему включені приклади використання мереж для прогнозування цін на фінансових ринках, приклади класифікації тварин і інші. Програмний пакет доволі складний та має багато надлишкової інформації. Окрім того, він великий за розміром, оскільки включає в себе модулі для створення різних типів нейромереж і не може бути використаний як окремий модуль.

Simbrain – кросплатформена система, написана на мові *Java* (рис. 1.7). Система орієнтована на візуальність і простоту. Включена реалізація двовимірного світу з різними об'єктами, для тестування нейронних мереж, які використовуються для управління.

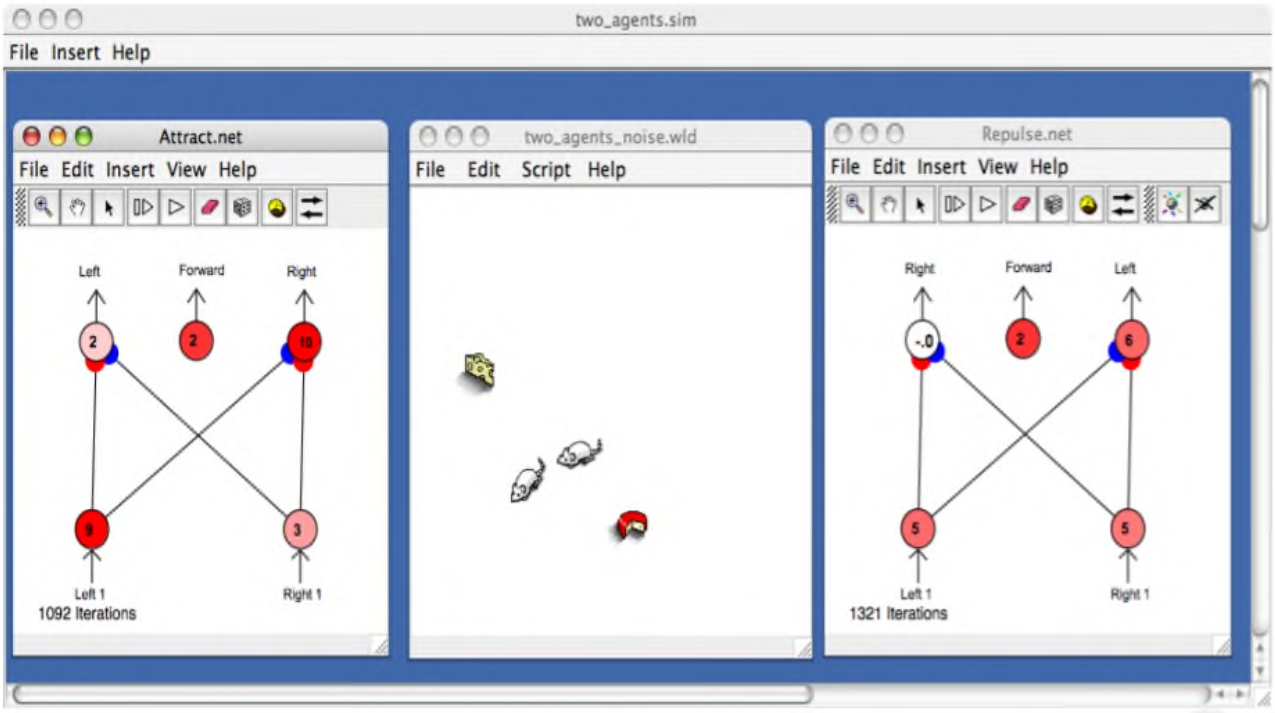

Рис. 1.7. Інтерфейс програми *Simbrain*

В системі реалізовано велику кількість різних варіантів нейронів і типів мереж, присутнє навчання без учителя. Створення та редагування нейронних мереж в середовищі *Simbrain* відбувається візуально. У разі візуального побудови, користувач може бачити процес зміни ваг в мережі, передачі сигналу. Можливо відстежувати і алгоритм зворотного поширення помилки. Але користувач також може скористатися *Java API* для конструювання власних завдань і додавання необхідних технологій в систему. Введення вибірок в мережу здійснюється вручну, що не завжди зручно. Ця система більше призначена для навчання, аніж для створення працюючого модуля. Складнощі з введенням навчальних даних, а також надлишковий розмір програми через наявність бібліотек для візуалізації роблять програму непідходящою для вирішення поставленої задачі.

MATLAB Deep Learning Toolbox (раніше *Neural Network Toolbox*) – це пакет розширення *MATLAB*, що містить засоби для проектування, моделювання, розробки та візуалізації нейронних мереж (рис. 1.8). Пакет забезпечує всебічну підтримку типових нейромережевих парадигм і має відкриту модульну архітектуру. Пакет містить функції командного рядка і графічний інтерфейс користувача для швидкого покрокового створення нейромереж. Крім цього *Deep Learning Toolbox* забезпечує

підтримку *Simulink*, що дозволяє моделювати нейромережі і створювати блоки на основі розроблених нейромережевих структур.

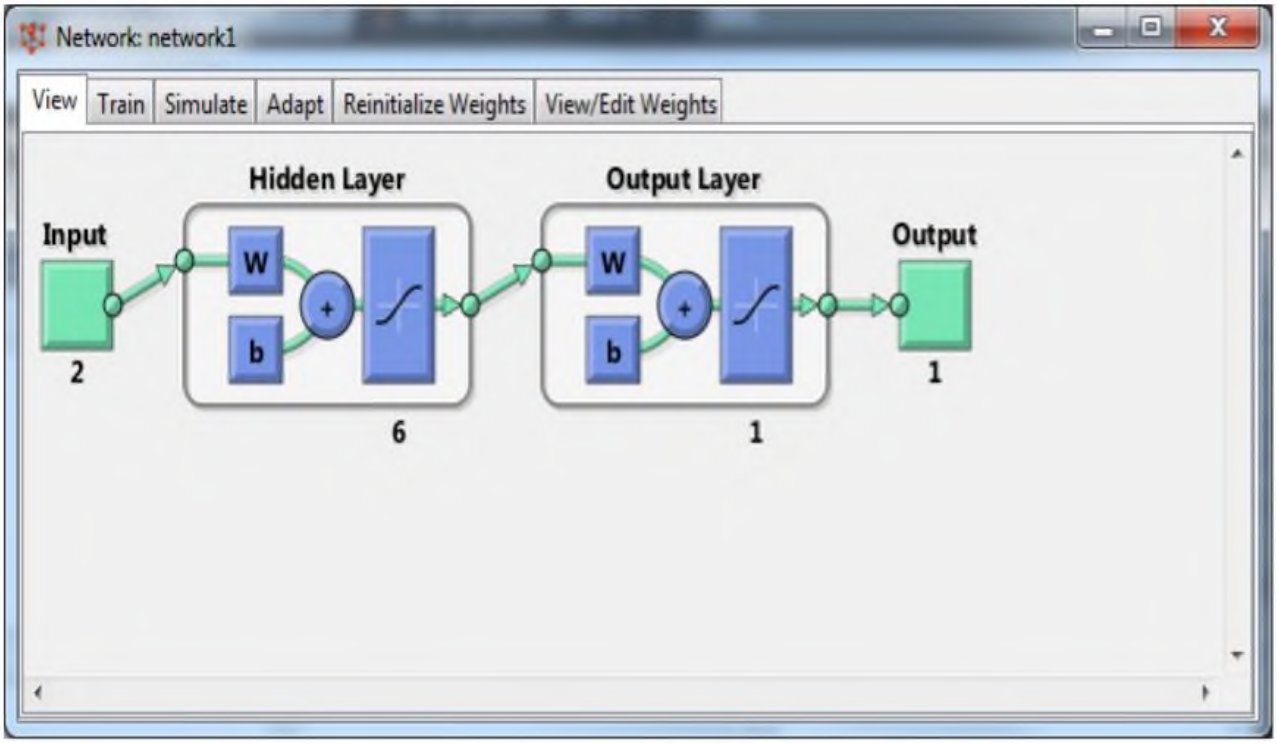

Рис. 1.8. Інтерфейс модуля *Deep Learning Toolbox* в *MATLAB*

Ключові характеристики модуля:

– графічний інтерфейс користувача для покрокового створення, навчання і імітаційного моделювання нейронних мереж;
– підтримка найбільш поширених керованих і некерованих мережевих структур;
– повний перелік навчальних і тестуючих функцій;
– динамічні алгоритми навчання мереж, що включають тимчасову затримку, нелінійну авторегресії (*NARX*), ланцюгові і настроюються динамічні структури;
– блоки *Simulink* для створення нейронних мереж і розвинених блоків для систем контролю;
– автоматична генерація блоків *Simulink* з об'єктів нейронної мережі;

– модульне подання мережі, що дозволяє створювати необмежену кількість вхідних шарів і об'єднаних мереж, а також графічне представлення архітектури мережі;
– функції попередньої і пост обробки і блоки *Simulink* для поліпшення процесу навчання і оцінки продуктивності мережі;
– візуалізація топології і процесу навчання нейронної мережі.

Як і попередні системи, дана система має надлишкові модулі, які збільшують розмір програми в цілому. Окрім того, хоча цей фреймворк підтримує більшу різноманітність нейромереж, це комерційний проект, отже він не безкоштовний.

Encog – система *Encog* є найбільш повною з розглянутих (рис.1.9). Вона включає в себе велику кількість ефективно написаних *Java* класів. На відміну від інших систем, в ній реалізовані не тільки нейромережеві алгоритми розпізнавання, але і кластеризація, генетичні алгоритми, приховані марковскі моделі, байєсовські мережі та ін. Серед підтримуваних архітектур нейронних мереж такі, як адалайн, машина Больцмана, нейронні мережі зворотного поширення, рекурентні мережі Ельмана, рекурентні мережі Джордана, самоорганізуються карти Кохонена і т.д. Доступні наступні функції активації нейронів: біполярна, змагальна, функція Еліотта, функція Гаусса, гіперболічний тангенс, лінійна, синусоїдальна, сігмоїда. У порівнянні з іншими, система *Encog* написана дуже ефективно, підтримується паралельне функціонування мереж на машинах з декількома процесорами.

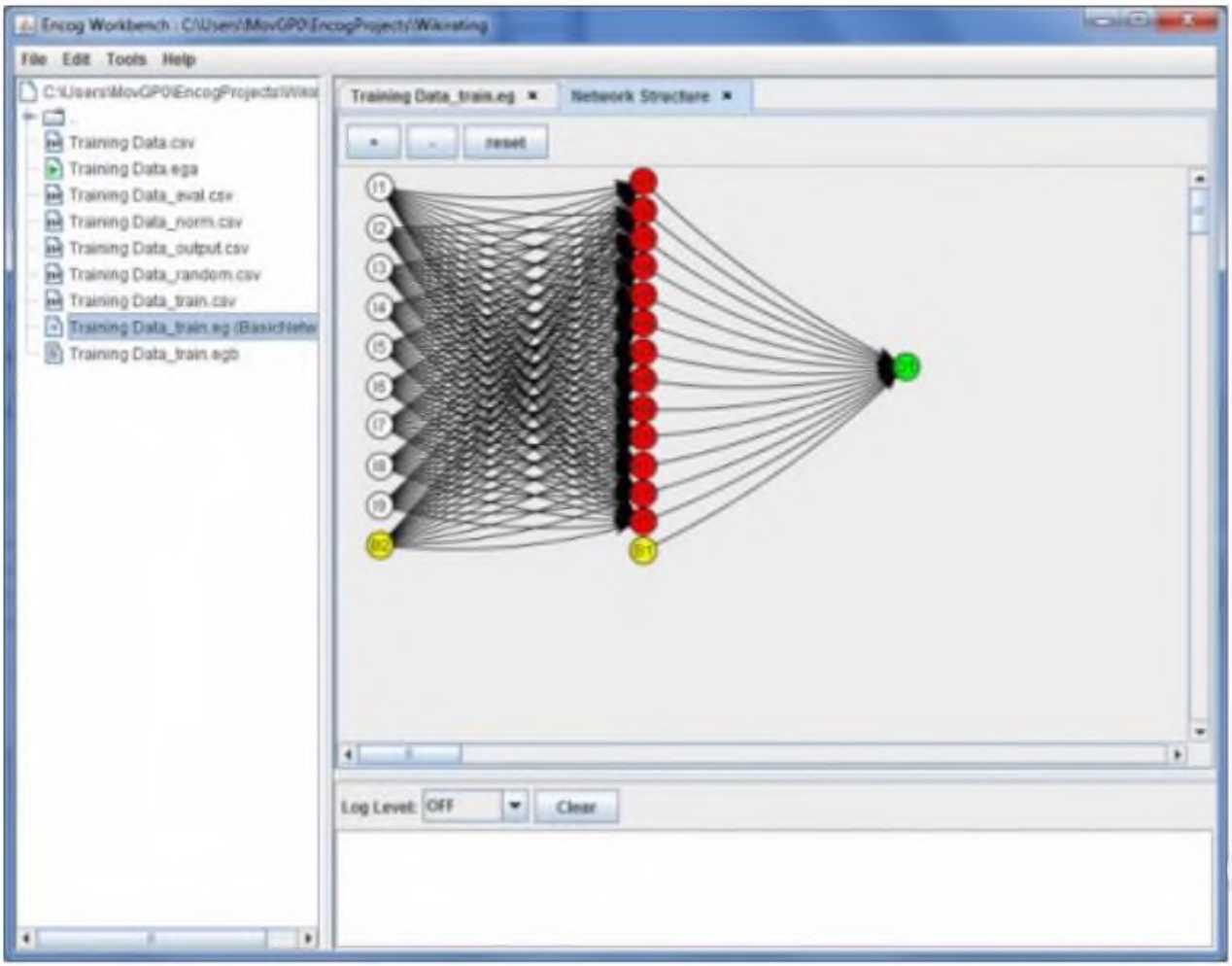

Рис. 1.9. Інтерфейс системи *Encog*

Усі перечисленні реалізації мають багато доступних топологій нейромереж, проте не дозволяють додавати нові архітектури мереж, змінювати характеристики навчання, оптимізувати структуру мережі. Системи широкого профілю, що працюють з різними завданнями і включають в себе велику кількість доступних архітектур нейронних мереж, не можуть безпосередньо застосовуватися до поставленої задачі.

1.7. Висновки до розділу

В першому розділі дипломної роботи було розглянуто основи автоматичних регуляторів. Автоматичні регулятори необхідні для стабілізації роботи систем від зовнішніх та внутрішніх впливів. Оскільки зовнішні впливи можуть бути різні за силою та напрямом, регулятор повинен вміти стабілізувати роботу системи незалежно від моменту початку роботи системи. Для нормальної роботи САР

необхідно підібрати оптимальні параметри регулятора, оскільки від вибору параметрів залежить, буде автоматичний регулятор стабілізувати роботу системи, чи навпаки, буде створювати ще більшу похибку. Для підбору параметрів можуть бути використані ШНМ. На даний момент існують нейромережі, які підбирають параметри автоматичного регулятора, проте такі системи можуть використовуватися лише для однієї задачі і здебільшого знаходяться в закритому доступі.

Розглянуто існуючі програмні системи для створення нейромереж, проте, більшість з них мають великий розмір і багато надлишкового функціоналу, а нейромережі, створені в таких програмах не можуть бути використані в окремих додатках.

РОЗДІЛ 2
ПРОЕКТУВАННЯ НЕЙРОМЕРЕЖЕВОГО ДОДАТКУ

2.1. Постановка задачі

Для прикладу візьмемо першу задачу з п’ятого підрозділу другого розділу методичного посібника [6]. Для того, щоб забезпечити водою жителів села, побудували водонапірну вежу, в яку насосом закачується вода з річки. Кожен споживач може в будь-який момент включити воду на своїй ділянці, наприклад, для поливу. Потрібно побудувати систему, яка автоматично підтримує заданий рівень h_0 води в цистерні (в метрах).

Для спрощення уявімо, що ця водонапірна вежа являє собою бак з водою. У нижній частині бака просвердлений отвір, через який витікає вода. Площу перетину бака позначимо через S. Дозволимо потоку витікаючої рідини q змінюватися незалежно (в теорії управління це називається навантаженням на об'єкт).

Будемо вважати, що споживачів досить багато, тому у когось завжди включена вода і насос постійно працює на закачування води в цистерну. Для управління рівнем води h ми можемо змінювати його потік Q (в $м^3/с$). Таким чином, рівень h – це регульована величина, а потік Q – сигнал управління. Для зворотного зв'язку використовуємо датчик, що вимірює рівень води h в цистерні (для простоти завдання не вдаватимемося в подробиці типу і швидкодії датчика, просто приймемо що датчик здійснює вимірювання кожну секунду, і коригування значення насоса так само відбувається раз в секунду).

Побудуємо математичну модель об'єкта, тобто цистерни. Потік на виході q (в $м^3/с$) показує, скільки води витікає з цистерни за 1 $с$ – це навантаження (рис.2.1).

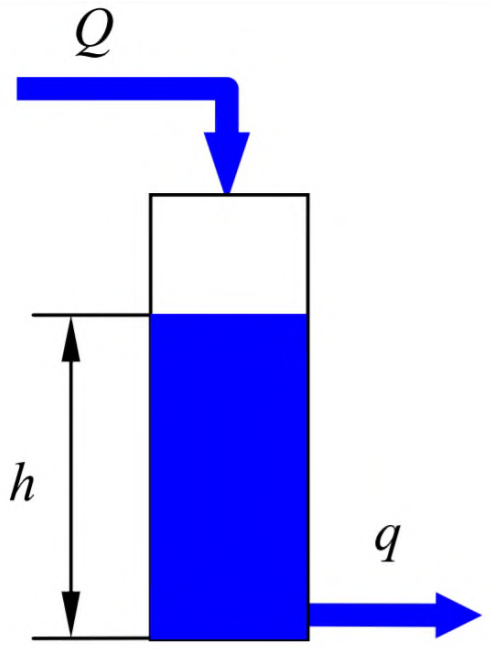

Рис 2.1. Графічне зображення цистерни

Зміна рівня Δh залежить від різниці потоків $Q - q$ і площі перетину цистерни S. Якщо різниця потоків постійна протягом інтервалу часу Δt, то

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

У загальному випадку потрібно використовувати інтеграл:

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

Нехай в момент часу $t = 0$ рівень води дорівнює заданому значенню, а вхідний і вихідний потоки рівні ($Q(0) = q(0) = q_0$), так що рівень не змінюється. Цей режим приймаємо за номінальний (робочу точку). Для того, щоб отримати рівняння у відхиленнях, уявімо потоки у вигляді

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

де $\Delta Q(t)$ – мале відхилення вхідного потоку від номінального режиму.

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

де $\Delta q(t)$ – мале відхилення вихідного потоку від номінального режиму.

Тоді, опускаючи знак збільшення Δ, можна записати модель об'єкта управління у формі:

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

де $h(t)$ – відхилення рівня води в цистерні від номінального значення;

$Q(t)$ – відхилення вхідного потоку від номінального значення;

$q(t)$ – відхилення вихідного потоку від номінального значення.

Для спрощення далі приймемо S = 1 *м*^{2}. Тобто, при навантажені q = 0.5 *м*^{3}*/с*, об’єм води в цистерні за одну секунду зменшиться на 0.5 *м*^{3}, отже висота стовпа води зменшиться на 0.5 *м*. В якості зворотного зв'язку ми будемо використовувати сигнал з датчика рівня. Помилка управління обчислюється як різниця між заданим і виміряним рівнями води:

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

Застосуємо найпростіший регулятор – підсилювач з коефіцієнтом K (або пропорційний регулятор, П-регулятор), який управляє потоком за формулою:

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

Структурна схема системи управління показана на рис. 2.2. Знак інтеграла означає ланку, модель якого – оператор інтегрування. За допомогою кружка з секторами позначається складання сигналів. Якщо якийсь сектор зафарбований чорним кольором, то сигнал, що входить в нього віднімається (враховується в сумі зі знаком «мінус»). Крім сигналів, про які вже йшлося, на малюнку показаний також шум вимірювання) $m(t)$, що спотворює показання датчика (в даній задачі будемо вважати, що шум відсутній, або настільки малий, що ним можна знехтувати).

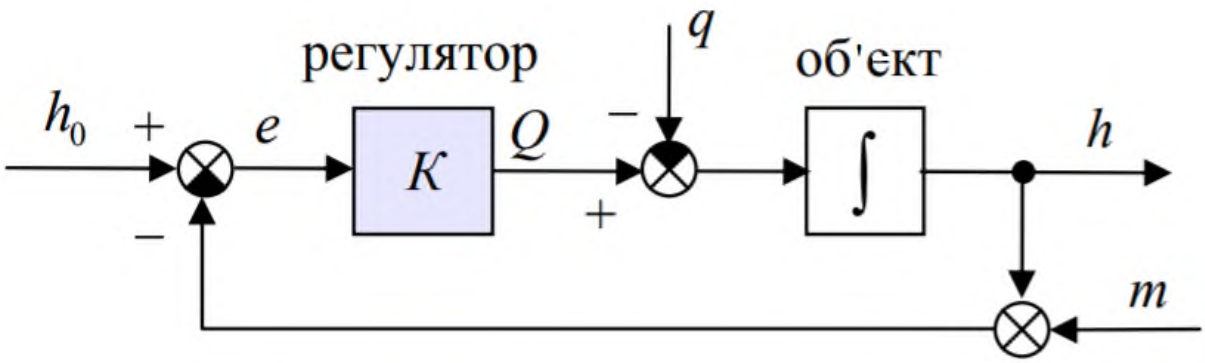

Рис. 2.2. Структурна схема системи управління

Для управління даною системою будемо змінювати коефіцієнт *K* в залежності від навантаження на систему і поточного значення подачі насоса.

Спочатку визначимося з вхідними та вихідними параметрами нейромережі.

Позначимо поточну ітерацію літерою *i*, тоді формула (2.6) набуває вигляду:

$$Q_i = K_i * [h_0 - h_i]. \tag{2.8}$$

Для спрощення керування, будемо визначати значення подачі насоса як суму значення подачі з попередньої ітерації та значення приросту:

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

де Q_i – необхідне значення подачі насоса на поточній ітерації;

Q_{i-1} – значення подачі насоса на попередній ітерації;

h_0 – бажана висота води;

h_i – висота води на поточній ітерації;

Знайдемо з даного виразу значення коефіцієнту *K*:

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

При постійній витраті в межі двох секунд, можна вважати, що

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

$$Q_i = q + (h_0 - h_i). \tag{2.12}$$

Після підстановки формул (2.10) і (2.11) у формулу (2.9) отримуємо наступний вираз:

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

Як видно з формули (2.13), конкретне значення витрати води, як і попереднє значення подачі насоса в даному випадку не важливо, оскільки для даного завдання досить визначити приріст насоса в порівнянні з попереднім значенням.

Для перевірки даної формули підставимо реальні дані в формулу і порахуємо *К* для довільного значення. Приймемо h_0 = 1 *м*, h_{i-1} = 0.8 *м*, h_i = 0.5 *м*.

В такому випадку підставивши дані у формулу (2.13) отримаємо:

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

Перевіримо коректність визначеного коефіцієнта. Для цього приймемо довільну витрату води, наприклад q = 0.4 *м³/с*. Враховуючи, що за одну ітерацію рівень води зменшився на 0.3 *м*, можна розрахувати подачу насоса на попередній ітерації як різницю між витратою і різницею висоти стовпів води на поточній і попередній ітераціях:

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

Подачу насоса , яку необхідно встановити, порахуємо за формулою (2.8):

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

Значення різниці висоти стовпа води в цистерні порахуємо як різницю між подачею та витратою води за одну секунду:

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

Отже, висота стовпа води в цистерні на наступній ітерації буде рівна сумі висоти стовпа води на поточній ітерації і різниці висоти стовпа води на наступній ітерації:

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

Як можна побачити, на наступній ітерації буде досягнуто бажаної висоти стовпу води в цистерні. Даний спосіб визначення очікуваного результату задовольняє вимогам задачі.

Важливо відмітити, що використання даного регулятора не дасть досягти позиції рівноваги. При використанні іншого типу регулятора, точність керування може бути вища, а статична похибка менша, або взагалі відсутня.

Після визначення завдання, необхідно визначитися з параметрами нейромережі, яка допоможе визначити регулятор для системи, математична модель якої описана рівняннями (2.1 – 2.13).

2.2. Визначення параметрів нейромережі

2.2.1. Визначення топології нейромережі

Штучні нейронні мережі надзвичайно різноманітні за конфігураціями. Незважаючи на це, мережеві парадигми мають багато спільного [23].

ШНМ розрізняють за топологічними типами відповідно до структури зв’язків між нейронами мережі. Різні топології нейромереж використовуються для вирішення різних задач.

Архітектури мереж та задачі, для яких вони використовуються, а також спосіб їх навчання наведено у 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

Фактично, задача полягає у апроксимації деякої функції. Для апроксимації функцій добре зарекомендувала себе штучна нейромережа прямого поширення – це вид нейронної мережі, в якій сигнали поширюються в одному напрямку, починаючи від вхідного шару нейронів, через приховані шари до вихідного шару і на вихідних нейронах отримується результат опрацювання сигналу. В мережах такого виду немає зворотних зв’язків.

Саме таку нейромережу вибираємо для реалізації задачі визначення параметрів регулятора.

2.2.2. Визначення кількості прошарків і нейронів нейромережі

Нейронна мережа складається з прошарків – вхідного, вихідного і прихованих. Обов'язковими є два шари – вхідний і вихідний. Кількість прихованих прошарків може бути довільна. Неможливо аналітично розрахувати кількість прошарків і кількість нейронів у кожному прошарку, які необхідно використовувати в ШНМ для рішення конкретної задачі прогнозного моделювання. Загального правила скільки повинно бути таких прошарків немає, зазвичай обирається від одного до трьох прихованих прошарків. Чим більш нелінійною є задача, тим більше прихованих прошарків повинно бути. Для вибору кількості прихованих прошарків можна використовувати один з наступних запропонованих способів:

– експериментально – кількість прошарків вибирається за допомогою систематичних експериментів з різною кількістю прошарків та нейронів на одному і тому ж наборі навчальних даних. На основі результатів експериментів можна вибрати підходящу кількість прошарків, базуючись на швидкості навчання, схильності до перенавчання, найменьшій похибці обчислення, тощо;
– інтуїтивно – параметри ШНМ можуть бути задані згідно інтуїції. Наприклад, згідно розуміння проблемної області, можна вважати, що для вирішення проблеми прогнозування потрібна глибока ієрархічна модель, або є позитивний досвід використання ієрархічної моделі для задачі схожого типу. У такому випадку необхідно вибрати конфігурацію ШНМ, яка має багато рівнів глибини;
– завжди використовувати велику кількість прошарків. Існує припущення, що використання глубоких ШНМ, які мають велику кількість прошарків, може бути еврістичним підходом до конфігурування мереж для вирішення задач прогнозного моделювання. Недоліком такого підходу є те, що чим більша кількість прошарків у нейромережі, тим повільніше вона навчається і тим більша навчальна вибірка необхідна для навчання;

– використовувати досвід попередників. Цей спосіб включає в себе пошук реалізованих нейромереж для подібних задач і, на основі результатів та певних показників якості таких нейромереж, вибір відповідної кількості прошарків;
– використовувати всі перечислені способи. Можна знайти реалізації нейромереж для схожих задач і на основі конфігурацій розроблених нейромереж вибрати кількість прошарків і нейронів. Після навчання ШНМ кількість прошарків та нейронів мережі можна збільшувати, або зменшувати, в залежності від певного показника якості. Тобто, якщо величина похибки прийнятна, а швидкість навчання дуже повільна, то можна зменшити кількість прошарків.

Для нейромережі що проектується, приймемо кількість прихованих прошарків рівною трьом. В більшості випадків цього повинно бути достатньо.

В штучній нейромережі прямого поширення, всі елементи попереднього прошарку зв'язані з усіма елементами наступного. Кількість нейронів у першому та останньому прошарках залежить від того, скільки полів вказано як вхідні та вихідні. Як видно з формули (2.13), вхідними даними нейромережі, що проектується, будуть параметри h, h_i і h_{i-1}, а вихідним значенням – K, коефіцієнт для даної ітерації. Тобто, вхідний прошарок нейромережі буде складатися з трьох нейронів, а вихідний прошарок – з одного.

Чіткого алгоритму для визначення кількості нейронів у прихованих шарах нейромережі – немає. Їх повинно бути не дуже мало, оскільки в такому випадку нейромережа буде погано навчатися, але їх також не повинно бути дуже багато, оскільки навчання такої мережі буде займати дуже багато часу і є вірогідність, що частина такої нейромережі не буде використовуватися при обрахуванні результатів. Окрім того, необхідно, щоб число зв'язків між нейронами було меншим за кількість прикладів в навчальній вибірці. Інакше нейромережа втратить здатність до узагальнення, а просто «запам'ятає» всі приклади з навчальної вибірки. Тоді при тестуванні на прикладах, наявних у навчальній вибірці вона буде демонструвати прекрасні результати, а на реальних даних – погані.

Можна вибрати більшу кількість нейронів, а потім поступово зменшувати їх кількість, доки не буде досягнуто бажаної точності за мінімальної кількості нейронів.

Виберемо кількість нейронів для першого прошарку – 60, для другого – 40, для третього – 20.

2.2.3. Визначення способу навчання нейромережі

Мережа навчається, щоб для деякої множини входів *X* давати бажану множину виходів *Y*. Кожна така вхідна (або вихідна) множина розглядається як вектор. Навчання здійснюється шляхом подання на входи множини значень функції, визначення вихідного значення, та корегування ваг зв'язків між нейронами за певним способом.

Навчання нейронних мереж можна представити як задачу оптимізації. Оцінити – означає вказати кількісно, добре чи погано мережа вирішує поставлені їй завдання. Для цього будується функція оцінки. Вона, як правило, явно залежить від вихідних сигналів мережі і неявно (через функціонування) — від всіх її параметрів. Найпростіший і найпоширеніший приклад оцінки — сума квадратів відстаней від вихідних сигналів мережі до їх необхідних значень. В даній дипломній роботі в якості функції оцінки буде використовуватися середньоквадратична похибка.

Існують три загальні парадигми навчання: «з вчителем», «без вчителя» (самонавчання) і змішана.

У першому випадку нейромережа має у своєму розпорядженні правильні відповіді (виходи мережі) на кожен вхідний приклад. Ваги налаштовуються так, щоб мережа виробляла відповіді як можна більш близькі до відомих правильних відповідей.

Навчання «без вчителя» не вимагає знання правильних відповідей на кожен приклад навчальної вибірки. У цьому випадку розкривається внутрішня структура даних та кореляція між зразками в навчальній множині, що дозволяє розподілити зразки по категоріях.

При змішаному навчанні частина ваг визначається за допомогою навчання зі вчителем, у той час як інша визначається за допомогою самонавчання.

В даній дипломній роботі використовується парадигма навчання «з вчителем», тобто генерується певна кількість даних для навчання, і завдяки порівнянням очікуваного та отриманого результатів, корегуються значення ваг у зв’язках між нейронами.

Для зміни ваг на входах нейрона, необхідно застосувати один с методів навчання. Один з найпоширеніших методів навчання – метод зворотного поширення помилки (рис. 2.3).

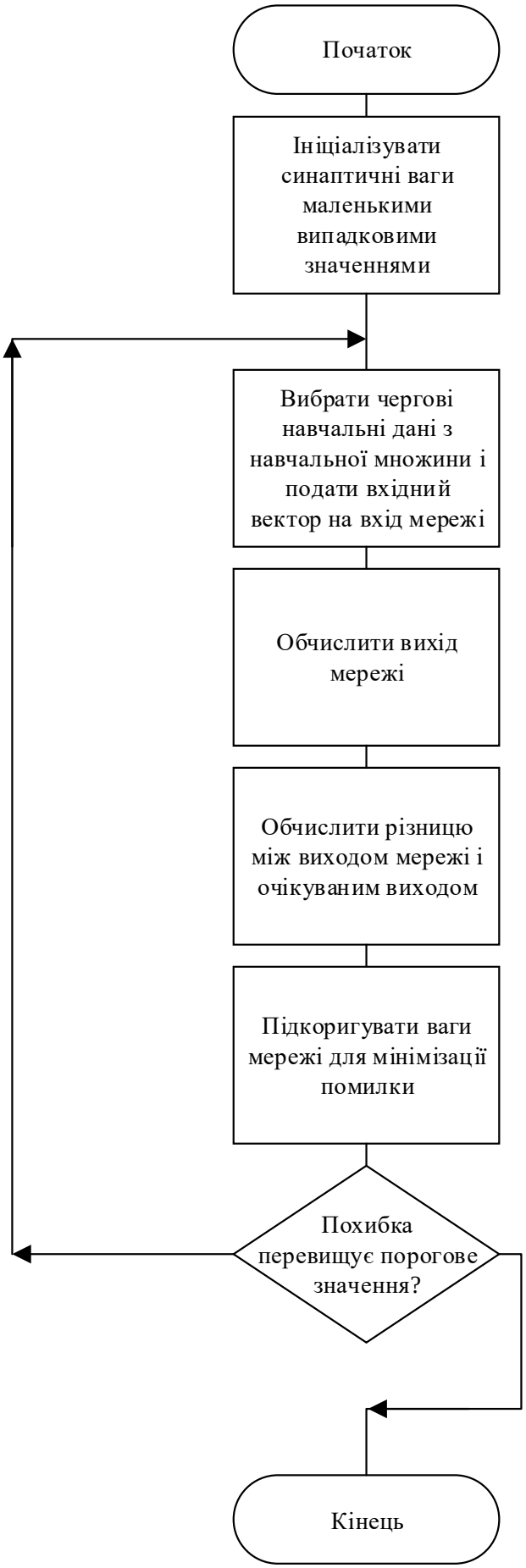

Рис. 2.3. Схема алгоритму зворотного поширення помилки

Це ітеративний градієнтний алгоритм, який використовується з метою мінімізації помилки роботи ШНМ та отримання бажаного виходу. Основна ідея цього методу полягає в поширенні сигналів помилки від виходів мережі до її входів, в напрямку, зворотному прямому поширенню сигналів у звичайному режимі роботи.

Алгоритм зворотного поширення помилки застосовується для багатошарового перцептрону. Для можливості застосування методу зворотного поширення помилки функція активації нейронів повинна бути диференційованою.

Ваги на входах нейронів змінюються за правилом «дельта», яке є одним з найбільш використовуваних. Це правило базується на простій ідеї неперервної зміни синоптичних ваг для зменшення різниці («дельта») між значенням бажаного та реального вихідного сигналу нейрона.

За цим правилом мінімізується середньоквадратична похибка мережі. Це правило також згадується як правило навчання Відрова-Хофа та правило навчання найменших середніх квадратів.

У правилі «дельта» похибка отримана у вихідному прошарку перетворюється похідною передатної функції і послідовно пошарово поширюється назад на попередні прошарки для корекції синоптичних ваг. Процес зворотного поширення похибок мережі триває до досягнення першого прошарку.

Незважаючи на численні успішні застосування алгоритму зворотного поширення помилки, він не є панацеєю. Найбільше неприємностей приносить невизначено довгий процес навчання. У складних завданнях для навчання мережі можуть знадобитися дні або навіть тижні, вона може і взагалі не навчитися. Основними причинами такої поведінка є:

– параліч мережі (у процесі навчання мережі значення ваг можуть стати дуже великими значеннями в результаті корекції, що може призвести до того, що всі, або більшість нейронів будуть функціонувати при дуже великих значеннях вихідного значення, в області, де похідна функції дуже мала, а отже процес навчання може практично завмерти);
– локальні мінімуми (оскільки метод зворотного поширення похибки використовує різновид градієнтного спуску, то в певний момент мережа

може потрапити в локальний мінімум і залинатися там, навіть якщо існують більш глибокі мінімуми);

– розмір кроку (якщо розмір кроку фіксований і дуже малий, то збіжність надто повільна, якщо ж він фіксований і занадто великий, то може виникнути параліч або постійна нестійкість).

2.2.4. Визначення функції сумування та активаційної функції

У наявних на цей час пакетах програм штучні нейрони називаються «елементами обробки» і мають набагато більше можливостей, ніж простий штучний нейрон. На рис. 2.4 зображена детальна схема спрощеного штучного нейрону.

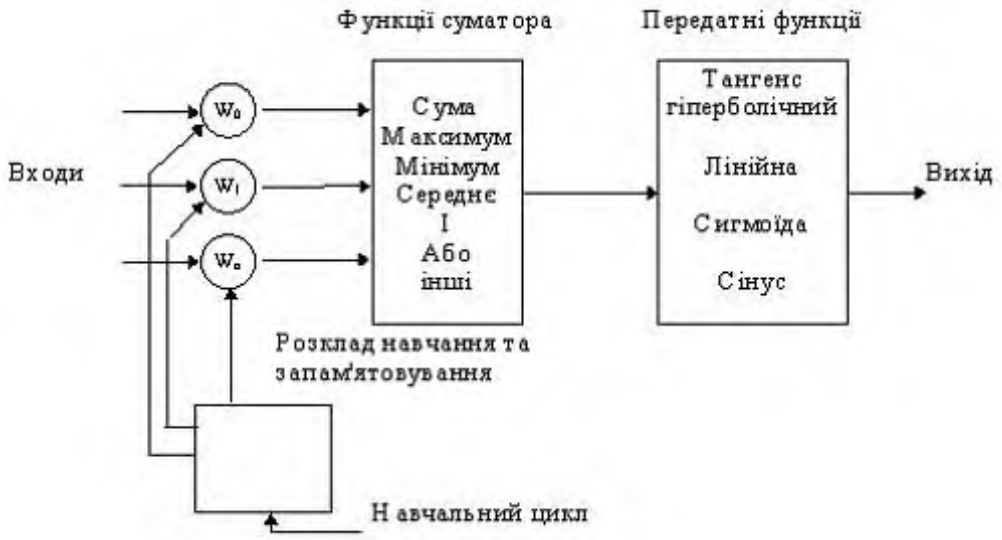

Рис. 2.4. Модель «елементу обробки»

Модифіковані входи передаються на функцію сумування, яка переважно тільки сумує добутки. Проте можна обрати багато різних операцій, такі як середнє, найбільше, найменше, *OR*, *AND*, тощо, які могли б виробляти деяку кількість різних значень. Окрім того, більшість комерційних програм дозволяють інженерам-програмістам створювати власні функції сумування за допомогою підпрограм, закодованих на мові високого рівня (*C*, *C++*, *Python*). Інколи функція сумування ускладнюється додаванням функції активації, яка дозволяє функції сумування оперувати в часі.

Для нейромережі, що розробляється, буде використовуватися функція сумування, яка є найбільш поширеною функцією для нейромереж прямого поширення.

Для кожного прошарку необхідно вказати тип активаційної функції. Активаційна функція (функція активації, передавальна функція) – залежність вихідного сигналу штучного нейрона від вхідного.

Метод зворотного поширення помилки вимагає, щоб функція активації була:

– нелінійна – коли передавальна функція нелінійна, то, як доведено, двошарова нейронна мережа є універсальною апроксимацією функцій. Тотожна передавальна функція не має такої властивості. Коли декілька шарів використовують тотожну передавальну функцію, тоді вся мережа еквівалентна одношаровій моделі;
– неперервно диференційована – ця властивість бажана для використання методів оптимізації, заснованих на градієнті;
– монотонна.

Вхідний сигнал (NET), як правило, перетворюється активаційною функцією F і дає вихідний нейронний сигнал $OUT = F(NET)$ [20]. Активаційна функція $F(NET)$ може бути:

1. Пороговою бінарною функцією (рис. 2.5).

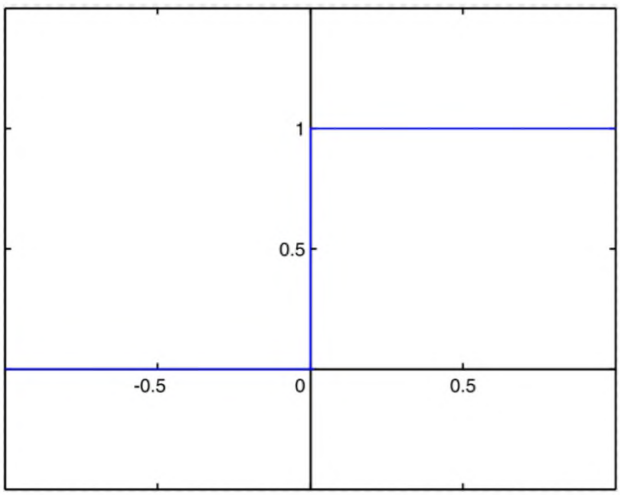

Рис. 2.5. Приклад графіку порогової бінарної функції

До тих пір, поки зважений сигнал на вході нейрона не досягає певного порогового рівня T – сигнал на виході дорівнює нулю. Як тільки сигнал на вході нейрона перевищує зазначений рівень, вихідний сигнал стрибкоподібно змінюється на одиницю. Власне, найперший представник багатошарових штучних нейронних мереж – персептрон – складався виключно з нейронів такого типу.

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

де T – деяка постійна порогова величина, або ж функція, що точніше моделює нелінійну передавальну характеристику біологічного нейрона.

2. Лінійною обмеженою функцією (рис. 2.6).

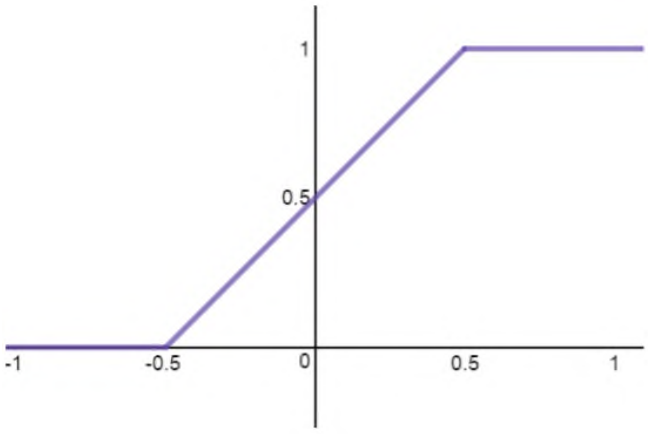

Рис. 2.6. Приклад графіку лінійної обмеженої функції

Тут присутні дві лінійні ділянки, в яких функція активації тотожно дорівнює максимально допустимому та мінімально допустимому значенню, і є ділянка, на якому функція строго монотонно зростає

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

4. Сігмоїдною (S-подібною) або логістичною функцією (рис. 2.7).

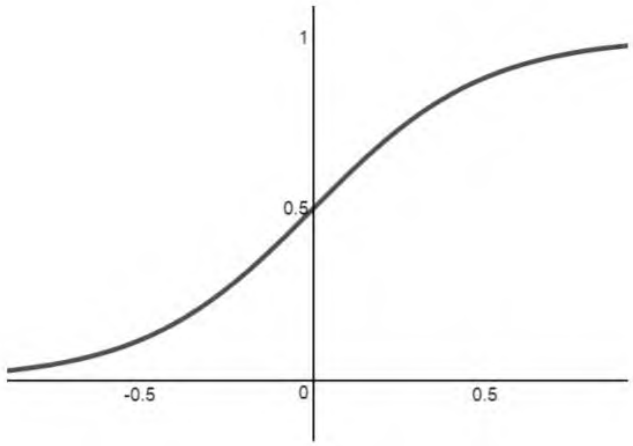

Рис. 2.7. Приклад графіку сігмоїдної функції

Обмеженість можливостей ШНМ із пороговою функцією активації нейронів призвела до використання функцій сігмоїдального типу При цьому активація нейрона відбувається плавно, що дозволяє перейти від бінарних виходів до аналогових. Особливість сігмоїдальних типів функцій в тому, що вони підсилюють слабкі сигнали, проте не насичуються від сильних.

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

3. Функцією гіперболічного тангенса (рис. 2.8).

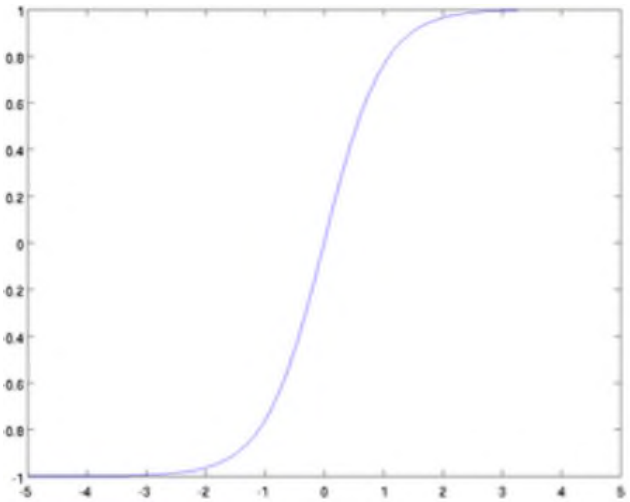

Рис. 2.8. Приклад графіку гіперболічного тангенса

Гіперболічний тангенс дуже схожий на сігмоїду, тому що це скоректована сігмоїдна функція. Тому така функція має ті самі характеристики, що і сігмоїда. Її

природа нелінійна, вона добре підходить для поєднання прошарків, область значених функцій в межах від мінус одиниці до одиниці. Тобто, активаційна функція не перевантажиться від великих значень. Однак варто відзначити, що градієнт тангенціальної функції більше, ніж у сігмоїдах (похідна більш нахилена). Рішення про те, використовувати сігмоїду або тангенс, повинно залежати від вимог до амплітуди градієнта.

$$OUT = th(C * NET). \tag{2.16}$$

5. Тотожною (лінійною) функцією (рис. 2.9).

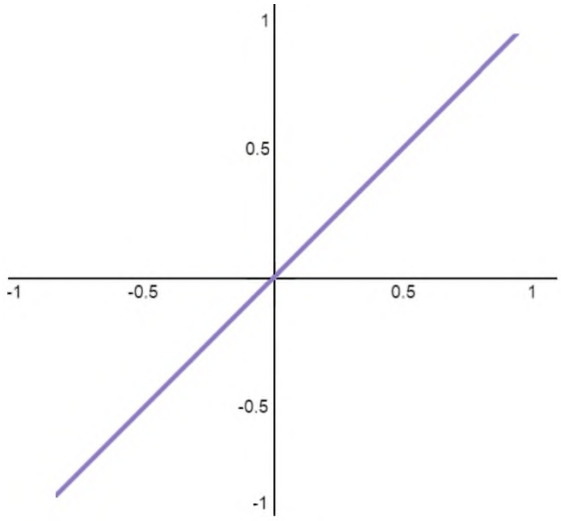

Рис. 2.9. Приклад тотожної функції

Сигнал на виході нейрона лінійно пов'язаний зі зваженою сумою сигналів на його вході. Іншими словами, на виході нейрона з такою активаційною функцією маємо те, що подавали на його вхід. У ШНМ з декількома прошарками, нейрони з даною активаційною функцією зазвичай застосовуються у вихідному або вхідному прошарках.

Дана активаційна функція застосовується доволі рідко, оскільки ШНМ, яка має в своєму складі тільки нейрони з лінійною активаційною функцією може виконувати дуже обмежений набір операцій (зміна знаку вхідних даних, сумування, тощо).

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

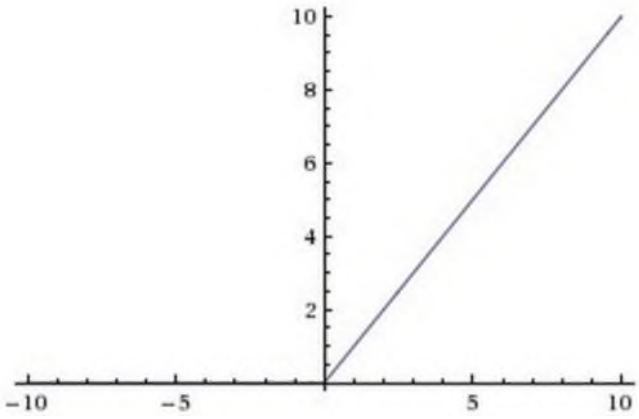

Рис. 2.10. Приклад *ReLu* функції

Важливою перевагою цієї функції перед сігмоїдальною та гіперболічним тангенсом є те, що обчислення значень цієї функції не є таким ресурсоємким, тож відбувається швидше.

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

Існує багато різних варіацій вищеподаних активаційних функцій, кожна з яких знаходить застосування в певних областях.

Вибір певної активаційної функції залежить від апроксимованої функції, та її області визначень. Наприклад, для визначення значень функції синуса зазвичай вибирають сігмоїдну функцію, або функцію гіперболічного тангенсу, оскільки вони за меншої кількості нейронів у шарі дають змогу отримати більшу точність роботи нейромережі.

Для нейромережі, що розробляється, буде застосовуватися параметрична випрямлена лінійна функція (*PReLu*), оскільки вона видає непогані результати, але на відміну від сігмоїда та тангенса – потребує меншу часу на вирахування похідної (що необхідно для навчання нейромережі) та обчислення результату функції. *PReLu* – це варіація *ReLu* функції.

ReLu нелінійна за своєю природою, а комбінація *ReLu* також нелінійна. Така функція є хорошим аппроксиматором, так як будь-яка функція може бути апроксимована комбінацією *ReLu*. Область допустимих значень *ReLu* – від нуля включно до нескінченності, тобто активація може «вибухнути».

Наступний пункт – розрідженість активації. При використанні великої нейронної мережі з безліччю нейронів, використання сігмоїд або гіперболічного тангенса буде тягти за собою активацію всіх нейронів в аналоговий спосіб. Це означає, що майже всі активації повинні бути оброблені для опису виходу мережі. Іншими словами, активація щільна, а це затратно.

В ідеалі необхідно, щоб деякі нейрони не були активовані, це зробило б активації розрідженими і ефективними. *ReLu* дозволяє це зробити. Нехай існує мережа із випадково ініціалізованими вагами (або нормалізованими), в якій приблизно 50% активацій майже рівні нулю через характеристик *ReLu* (повертає 0.1 для від'ємних значень x). У такій мережі включається менша кількість нейронів (розріджена активація), а сама мережа стає легшою [19].

Проте, у цієї функції активації також є і мінуси. Через те, що частина *ReLu* представляє собою горизонтальну лінію (для негативних значень X), градієнт на цій частині дорівнює 0. Через нульове значення градієнта, ваги не будуть коригуватися під час спуску. Це означає, що нейрони, які перебувають в такому стані, не реагуватимуть на зміни в помилці / вхідних даних (просто тому, що градієнт дорівнює нулю, нічого не буде змінюватися). Таке явище називається проблемою вмираючого *ReLu* (*Dying ReLu problem*). Через цю проблему деякі нейрони просто вимикаються і не будуть відповідати, роблячи значну частину нейромережі пасивної.

Однак існують варіації *ReLu*, які допомагають цю проблему уникнути. Наприклад, має сенс замінити горизонтальну частину функції на лінійну. Якщо вираз для лінійної функції задається виразом $y = 0.01x$ для області $x < 0$, лінія злегка відхиляється від горизонтального положення. Існує й інші способи уникнути нульового градієнта. Основна ідея тут – зробити градієнт не рівним нулю і поступово відновлювати його під час тренування.

Для даної нейромережі буде використано *PReLu* з параметром 0.01.

2.3. Висновки до розділу

В другому розділі дипломної роботи було розглянуто першу задачу, для якої необхідно розробити нейромережевий додаток для визначення параметрів автоматичного регулятора. Розглянуто формули, за якими можна сформувати набір навчальних даних. Розглянуто існуючі топології ШНМ, перераховано існуючі методи навчання для різних топологій, типи активаційних та сумуючих функцій та типи нейронів. Після аналізу вищевказаних даних було вибрано топологію ШНМ та основні параметри, такі як: функція суматора, активаційна функція, метод навчання, кількість прошарків та нейронів ШНМ.

РОЗДІЛ 3
РОЗРОБКА НЕЙРОМЕРЕЖЕВОГО ДОДАТКУ ДЛЯ ВИЗНАЧЕННЯ ПАРАМЕТРІВ АВТОМАТИЧНОГО РЕГУЛЯТОРА

3.1. Розробка програмного коду

3.1.1. Користувацький інтерфейс

Дана програма призначена для створення нейромережевих додатків, отже користувацький інтерфейс може бути простим. Основна задача інтерфейсу – дати змогу користувачу вибирати конфігурацію ШНМ, мати змогу змінити налаштування параметрів навчання нейромережі та дати можливість додавати файли з навчальною та тестовою вибіркою даних.

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

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

Рис. 3.1. Опції командного рядку користувацького інтерфейсу

Тут користувач має змогу вибрати одну з опцій: ініціалізувати існуючу нейромережу, попередньо збережену в окремий файл, створити нову нейромережу, або закінчити роботу з програмою.

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();
```

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

В залежності від вибору користувача викликається відповідний метод – ініціалізація нейромережі з файлу у методі *getExistingNetwork*, або створення нової нейромережі у 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());
```

```
                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 нейромережі з файлу, або створення ШНМ з вказанням необхідних параметрів, користувачу пропонується вибрати одну з опцій (рис. 3.2):

– почати навчання нейромережі (процес та логіка навчання буде описана далі);
– виконати обчислення (з’явиться запит ввести вхідні дані і після розрахунку у консоль виведеться результат обчислення);
– зберегти нейромережу у файл;
– вибрати іншу нейромережу (користувача поверне до попередніх опцій вибору);
– змінити налаштування, або вивести інформацію про нейромережу у 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
```

Рис. 3.2. Доступні опції для роботи з нейромережею

```
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
        }
    }
}
```

Рис. 3.3. Приклад виводу інформації про нейромережу у консоль

Після виводу інформації про ШНМ у консоль користувач може перевірити кількість прошарків, кількість нейронів у прошарках, їх активаційні функції та ваги зв’язків між нейронами.

Взаємодія користувача з нейромережею в залежності від вибраної опції описана у методі *handleUserInteraction*.

```
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;
```

```
        }
    }
```

При виборі опції зміни параметрів навчання ШНМ, користувач має змогу змінити наступні параметри (рис. 3.4):

– змінити крок похибки (величина, на яку буде множитися похибка нейромережі при перерахуванні ваг зв'язків нейронів);
– змінити проміжок епох між перевірками правильності навчання ШНМ на тестовому наборі даних (користувацький інтерфейс зчитує файл з тестовою вибіркою, подає на вхід нейромережі вхідні дані і звіряє вихідний результат з очікуваним, вираховуючи середньоквадратичну похибку, яку після вирахування виводить в консоль);
– змінити кількість епох (кількість ітерацій навчання нейромережі, де одна ітерація відповідає одному проходу по всьому набору даних навчальної вибірки);
– змінити прийнятну похибку (якщо середньоквадратична похибка рівна цьому значенню – нейромережа припиняю навчання);
– змінити періодичність (в епохах) автоматичного зберігання ШНМ у файл (у випадку, якщо станеться якась помилка, час, затрачений на навчання нейромережі не буде марним);
– змінити періодичність (в епохах) видалення зв'язків та нейронів, що не приймають участі 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
```

Рис. 3.4. Можливі опції зміни параметрів навчання нейромережі

3.1.2. Прошарки та нейрони

На основі інформації з попередніх розділів необхідно розробити код програми для навчання і обробки даних ШНМ. Згідно з принципами об’єктно-орієнтованого програмування, хорошим рішенням буде розділити реалізацію програми на декілька модулів. Основним модулем є нейромережа, яка представлена окремим класом мови програмування *Java*. Вона містить прошарки, інформацію про кількість нейронів в прошарках, та інформацію про активаційні функції в прошарках. При ініціалізації нейромережі, дані передаються в контролер класу, де ініціалізуються прошарки.

Кожен прошарок містить посилання на нейрони попереднього прошарку (окрім першого прошарку), власні нейрони, та активаційну функцію.

В залежності від того, чи являється прошарок вхідним, визивається одна з функцій – *initInputNeurons*, або *initNeurons*, які ініціалізують вхідні нейрони, вихідні, або нейрони прихованих шарів.

```
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));
    }
  }
```

Вхідний та вихідний нейрони являються окремими випадками реалізації нейрона.

Важливою відмінністю вхідного нейрона є те, що у нього немає суматора і активаційної функції. Тобто, на виході такого нейрона буде те, що передано 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 прихованих прошарків однакові і відрізняються лише кількістю вхідних зв’язків та активаційною функцією. Вхідне значення такого нейрона вираховується як сума значень вхідних нейронів, помножено на відповідну вагу вхідного зв’язку (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());
    }
}
```

Сума вагів зв’язків використовується в подальшому для обрахування помилки нейрона.

Вихідне значення нейрона обчислюється в залежності від вибраної активаційної функції.

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

3.1.3. Метод зворотного поширення похибки

Окрім функцій, вказаних у попередньому підрозділі, нейрон містить функцію для обрахування похибки і налаштування вагів відповідним чином.

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

Метод зворотного поширення помилки для визначення значення, на яке необхідно зменшити, або збільшити вагу зв’язку, використовує похідну від активаційної функції.

```
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 вплинуло на вихідний результат нейромережі. Чим більше вплив конкретного нейрона, тим більшим буде значення похибки для цього нейрона.

На основі похибки конкретного нейрона вираховується похибка вхідного нейрона. Далі корегування відбувається для кожного вхідного нейрона, тобто для нейронів з попереднього прошарку. Існує вірогідність, що в певний момент часу, вага певного зв'язка може стати дуже великим, або дуже маленьким значенням (або навіть стати нескінченністю при перевищенні області допустимих значень числа з плаваючою точкою). Для вирішення даної проблеми вага такого зв'язка ініціалізується випадковим значенням.

```
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;
    }
}
```

Зв'язок, який певний час не використовується в обчисленнях – видаляється з нейромережі. Нейрон, у якого всі вхідні зв'язки видалені, також видаляється з нейромережі, оскільки такий нейрон жодним чином не впливає на вихідний результат нейромережі.

Швидкість зміни ваш зв’язків визначається змінною *PENALTY_STEP*, яку користувач може вказувати при навчанні нейромережі.

Дана змінна визначає величину кроку градієнтного спуску и при дуже великому значенні може виникнути ситуація, коли оптимальне рішення може просковзнути повз глобального мінімуму і більше ніколи туди не повернутися. При дуже маленькому значенні, система може застрягти в локальному мінімумі і найоптимальніше рішення також не буде досягнуто. Рекомендується починати з маленького значення коефіцієнта похибки і при необхідності збільшувати його під час навчання, якщо подальше навчання не призводить до зменшення похибки.

3.1.4. Активаційна функція

Активаційна функція повинна вираховувати вихідне значення нейрона і приймати участь у визначені похибки обчислення. Ці дві функції поєднуються в одному інтерфейсі *ActivationFunction.*

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

Інтерфейс лише визначає, які методи повинні бути реалізовані в класі активаційної функції. Конкретна реалізація методів повинна бути створена у відповідному класі активаційної функції згідно формул (2.13 – 2.17).

Наприклад, методи для реалізації активаційної функції *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 для реалізації активаційної тотожної функції:

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

З цих трьох прикладів можна побачити різницю між часом, необхідним для обчислення значення та похідної кожної з функцій – сігмоїда використовує більш складні арифметичні операції, а отже час, затрачений на обчислення значення та похідної, буде більший. Тотожна активаційна функція, в свою чергу, повертає значення параметра, а похідна чітко визначена і рівна одиниці, отже час обчислення значення та похідної, при використанні такої функції, буде найменшим. І хоча різниця не дуже велика, при великій кількості нейронів, вибір активаційної функції має значний вплив на швидкість навчання нейромережі.

3.1.5. Розробка функцій для навчання нейромережі

Основним методом для визначення значення на виході нейромережі 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;
}
```

Даний метод викликає метод *getCalculatedOutput*() у нейронів вихідного шару, які в свою чергу рекурсивно викликають розрахунок значень нейронів у попередніх шарах.

Метод *train*() є основним методом для навчання нейромережі. Як параметр в нього передається файл, з якого необхідно зчитувати дані. Метод построково зчитує дані, переводить їх у зручний для роботи формат і передає ці перетворені дані у 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 на вхід дані з файлу, перетворені у зручний для роботи формат і передає ці дані для розрахунку нейромережею. Після отримання результату роботи нейромережі, дані передаються у метод *adjustPenalties*(), який розраховує похибку для кожного вихідного нейрону і викликає метод для зміни ваг зв'язків між нейронами. Зміна ваг відбувається пошарово, від вихідного шару до вхідного (якщо вихідних нейронів декілька, то похибка обчислюється пошарово 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 розрізняється в залежності від вибраної активаційної функції прошарку.

Перевірка роботи ШНМ та визначення середньоквадратичної похибки виконується у методі *getError*(), який зчитує тестові дані з файлу, подає вектор вхідних даних на вхід нейромережі і на основі отриманих даних вираховує середньоквадратичну похибку.

```
    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 похибки менше за допустиме – нейромережа припиняє навчання. Якщо значення похибки не змінюється на протязі двох перевірок похибки – нейромережа припиняє навчання і пропонує користувачу змінити значення величини кроку для градієнтного спуску.

Окремої уваги заслуговує метод для видалення зв’язків, які не приймають участі у обраховуванні результата. Окрім того, якщо у нейрона немає вхідних зв’язків, то він також вважається таким, що не впливає на результат нейромережі і підлягає видаленню. Метод *deleteZeroLinksAndNeurons*() поступово, для кожного шару, видаляє зв’язки у нейронів, а потім ще раз проходить по цим нейронам для видалення тих, що не мають в своєму складі жодного зв’язку.

```
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 у окремий файл для подальшого використання як модуль в будь-якій іншій програмі.

Частковий лістинг коду наведено у Додатку А.

3.2. Підготовка тестових даних

Підготовка тестових даних – дуже важливий етап створення нейромережі. Від якості і кількості навчальної вибірки залежить швидкість навчання нейромережі, здатність до перенавчання та здатність нейромережі до обчислення значень які знаходяться поза межами області значень навчальної вибірки.

Перед початком навчання нейромережі, згенеруємо набір тестових даних, щоб нейромережа могла зрозуміти, які значення повинні бути на виході мережі. Для цього напишемо клас, який буде реалізовувати дану функцію (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;
    }
}
```

Рис. 3.5. Код класу для реалізації заданої функції

Після генерації даних на основі створеного класу, отримано наступні навчальні дані (рис. 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|4.642093$3.7385979
4.0|1.1257081|3.4843764$-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|0.1896822$2.4278667
5.0|2.5305576|4.589836$-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|4.89965$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
```

Рис. 3.6. Частина навчальних даних

Для зручності оперування даними комп’ютером, дані розташовані в певному вигляді: «перший вхідний параметр» | «другий вхідний параметр» | … | «останній вхідний параметр» $ «перший вихідний параметр» | «другий вихідний параметр» | … |«останній вихідний параметр».

Кожен набір даних починається з нового рядку.

Перевіримо роботу програми для генерування навчальних даних. Для цього можна взяти один набір даних і підставити у формулу (2.12).

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

Приймемо довільну витрату води, наприклад q = 0.4 *м*^{3}/*с*. Враховуючи, що за одну ітерацію рівень води зменшився на 0.3 *м*, можна розрахувати подачу насоса на попередній ітерації як різницю між витратою і різницею висоти стовпів води на поточній і попередній ітераціях:

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

Подачу насоса , яку необхідно встановити, порахуємо за формулою (2.8):

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

Значення різниці висоти стовпа води в цистерні порахуємо як різницю між подачею та витратою води за одну секунду:

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

Отже, висота стовпа води в цистерні на наступній ітерації буде рівна сумі висоти стовпа води на поточній ітерації і різниці висоти стовпа води на наступній ітерації:

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

Отримано очікуваний результат.

3.3. Навчання нейромережі

Наступним кроком після генерації навчальних і тестових даних є навчання нейромережі. Оскільки процес визначення параметрів нейромережі не формалізований, то доволі легко зробити помилку при розробці нейромережі, що призведе до доволі тривалого часу навчання нейромережі, або взагалі її

неспроможності до навчання. Створимо нейромережу відповідно до параметрів, зазначених в другому розділі дипломної роботи (рис. 3.7).

```
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
```

Рис. 3.7. Визначення параметрів нейромережі

Параметри навчання нейромережі вказані на рис. 3.8.

```
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
```

Рис. 3.8. Параметри навчання нейромережі

На початку навчання нейромережі значення середньоквадратичної похибки доволі велике, проте поступово, з більшою кількістю ітерацій та зміною кроку градієнтного спуску отримано прийнятне значення похибки 0.01 (рис. 3.9). Подальша зміна параметрів навчання не призводить до значних змін (однак можна змінити кількість шарів, нейронів та активаційну функцію для отримання іншого результату), отже навчання можна закінчити.

```
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.
```

Рис. 3.9. Результат навчання нейромережі

3.4. Перевірка роботи нейромережевого додатку

Перевіримо нейромережу на одній вибірці з тестового набору даних та на даних, яких не було в тестовій вибірці (рис. 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
```

Рис. 3.10. Перевірка роботи нейромережі

Як видно з прикладу, нейромережа добре справляється з даними із навчальної вибірки та довільними значеннями. Проте, слід мати на увазі, що нейромережа навчалася на даних, отриманих при значенні витрати води від 1 до 4. Для значень менше, або більше вказаних, нейромережа не проходила навчання і буде видавати некоректні результати, і чим більше відхилення значення – тим більша буде похибка.

3.5. Розв’язання задачі про вибір одного з заданого набору регуляторів

Використаємо програму для створення нейромережі, яка буде вирішувати наступну задачу. Є результати моделювання руху динамічної системи, яку відхиляють від стану рівноваги, з кількома регуляторами, що мають різні параметри. Задачею регулювання є повернення системи до стану рівноваги. Кожна з отриманих траєкторій руху оцінюється певним значенням критерію якості. Таким чином, порівнюючи значення критерію якості, отримані при використанні кожного з регуляторів, ми можемо обрати той регулятор, який для даних початкових умов руху дає найкращий показник якості перехідного процесу.

Для даної задачі будемо використовувати як критерій якості час регулювання – це проміжок часу від моменту надходження на вхід ступінчастого впливу (завдання, збурення) до моменту, коли відхилення регульованої величини від заданого значення стає меншим деякого відносно малого числа δ (яке можна вважати зоною нечутливості регулятора). Так, зокрема, в задачах керування динамічними об’єктами прийнято вважати, що перехідний процес закінчиться в той момент часу, починаючи з якого, відхилення регульованої величини складає не більше ніж 5% від значення початкового відхилення (так званий 5%-й коридор). Отже, для даної задачі в якості показника якості вибираємо тривалість перехідного процесу до моменту часу, починаючи з якого, відхилення системи від стану рівноваги не перевищує 5% від початкового відхилення.

Для моделювання було використано динамічну систему, що описується наступним диференційним рівнянням:

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

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).

Програмою моделювання було згенеровано траєкторії та отримано значення критерію якості для дискретного набору початкових відхилень та для п’яти різних регуляторів. Для кожної траєкторії було розраховано відповідне значення критерію якості.

Перед нейромережевим додатком ставиться задача класифікації точок фазової площини початкових відхилень. Маючи навчальні дані про те, які значення критерію якості дає кожен з регуляторів у дискретному наборі початкових відхилень, ШНМ має визначити, який регулятор виявиться найкращим для довільної точки фазової площини.

На вхід ШНМ, що буде створена, подаємо координати точки початкового відхилення на фазовій площині, а на виходах повинні бути сигнали, які інтерпретуються як передбачені рівні доцільності використання кожного регулятора для даного відхилення. На основі цієї ШНМ необхідно буде створити додаток, який на основі отриманих результатів від нейромережі буде видавати номер регулятора, який доцільно використовувати для даного відхилення.

На рис. 3.11 подано 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
```

Рис. 3.11. Навчальна вибірка

На рис. 3.7 перше значення – це показник критерію якості, друге і третє значення представляють собою координати точки на фазовій площині (область

значень від мінус десяти до десяти включно), а четверте значення – номер регулятора, для якого при заданій точці обчислено показник критерію якості. Було використано 12500 значень у навчальній вибірці.

Після перетворення навчальної вибірки до вигляду, сприйнятного для нейромережі, отримуємо остаточну навчальну вибірку (рис. 3.12).

```
1  -4.69388|1.42857$0|0|1|0|0
2  -10.00000|-5.91837$0|0|0|1|0
3  8.77551|-7.14286$0|1|0|0|0
4  -8.77551|-5.91837$0|0|0|1|0
5  1.02041|8.36735$0|1|0|0|0
6  5.10204|7.95918$1|0|0|0|0
7  -8.36735|-5.51020$0|0|0|1|0
8  -5.10204|-0.61224$0|0|0|0|1
9  6.73469|-9.18367$0|0|0|1|0
10 -6.73469|-7.95918$0|0|1|0|0
11 7.55102|-8.77551$0|0|0|1|0
12 9.18367|2.24490$0|1|0|0|0
13 -2.24490|-3.06122$0|0|1|0|0
14 3.06122|-2.24490$0|1|0|0|0
15 0.20408|2.65306$0|1|0|0|0
16 2.65306|0.61224$0|1|0|0|0
```

Рис. 3.12. Перетворена навчальна вибірка

Тут перше та друге значення – це вхідні дані нейромережі (координати точки на фазовій площині), а останні п’ять значень – бінарне представлення виходів нейромережі, тобто номери регуляторів, які необхідно використати щоб перехідний процес був найшвидший. Оскільки при підготуванні навчальної вибірки для однієї точки вибирався регулятор з найкращим значенням критерія якості з поміж п’яти, то в навчальній вибірці отримуємо 2500 різноманітних даних (важливо, щоб дані розміщувалися невпорядковано).

Для даної задачі приймемо кількість прихованих прошарків – один, з кількістю нейронів – 50, та сігмоїдною активаційною функцією (приклад задання параметрів нейромережі наведено на рис. 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
```

Рис. 3.13. Параметри нейромережі

Для навчання нейромережі будемо використовувати ті самі налаштування, що і для попередньої нейромережі (див. рис. 3.8).

Час навчання даної нейромережі вийшов менший, аніж час навчання попередньої нейромережі, оскільки, незважаючи на тип активаційної функції, в даній нейромережі кількість шарів і нейронів у них набагато менша. Результат роботи нейромережі для першої точки з навчальної вибірки представлено на рис. 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
```

Рис. 3.14. Результат обчислення нейромережі

Як видно з результату, найбільш відповідний регулятор для заданої точки фазової площини – регулятор під номером три.

Оскільки дана нейромережа повинна давати відповідь на питання, який регулятор необхідно використовувати, то додамо нейромережу в нейромережевий модуль, який буде виводити номер регулятора як результат.

```
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<FloatDataWrapper> inputData = new
ArrayList<>();
                    inputData.add(firstNumber);
                    inputData.add(secondNumber);
                    network.setInputDataWrappers(inputData);
                    ArrayList<FloatDataWrapper> results =
network.processData();
                    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 +
1));
                } catch (Exception e) {
                    System.out.println("Unhandled exception: " + e);
                }
            }
        }
```

Результати роботи даного коду представлено на рис. 3.15.

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

Рис. 3.15. Результат роботи нейромережевого додатку

Для даного прикладу, координати точки вводяться користувачем вручну, проте у реальній системі, ці значення будуть передаватися в метод і результат обчислення нейромережі буде передаватися назад в систему для прийняття рішення щодо використання того, чи іншого регулятора.

3.6. Висновки до розділу

В третьому розділі описано створення та функціонування програми для генерації нейромережі заданої топології та методу навчання. Створена нейромережа для визначення параметрів автоматичного регулятора для задачі про керування водонапірною баштою. Описано підготовку даних для вказаної нейромережі та проведено навчання ШНМ методом зворотного поширення похибки.

Також розглянуто розв’язання задачі про вибір одного з заданого набору автоматичних регуляторів, яку подано як задача класифікації. Для розв’язання цієї задачі була створена та навчена друга ШНМ з іншою кількістю прошарків і активаційною функцією для вирішення задачі вибору регулятора модельної одномірної динамічної системи на основі показників критерію якості.

Проведено перевірку створених ШНМ на тестовому наборі даних. В результаті перевірки виявлено, що обидві нейромережі дають задовольні вирішення своїх задач.

ВИСНОВКИ

У результаті виконання дипломної роботи було створено програму для генерації нейромереж та нейромережевий додаток визначення параметрів автоматичного регулятора.

В першому розділі дипломної роботи було проведено дослідження актуальності проблеми застосування нейромереж для керування автоматичними регуляторами. Провівши дослідження, можна стверджувати, що визначення параметрів за допомогою нейромереж на даний момент є одним з найперспективніших напрямків розвитку автоматичних регуляторів. Для таких регуляторів достатньо отримати деякі експериментальні дані, які можуть бути використані для навчання нейромережі.

Враховуючи складність та неточність визначення параметрів автоматичного регулятора за допомогою традиційних методів Циглера-Нікольса, формульного метода визначення налаштувань регулятора, налаштування регуляторів за номограмами та розрахунку налаштувань за частотними характеристиками об'єкта (забезпечення заданого запасу стійкості в системі) [2], використання нейромереж, на мою думку, є найбільш доцільним та простим рішенням.

У другому розділі дипломної роботи розглядається задача, яку необхідно вирішити за допомогою нейромережі. Для створення нейромережевого додатку була обрана задача керування системою водопостачання з теорії автоматичного управління.

Для вирішення поставленого завдання була вибрана топологія ШНМ та її основні параметри, такі як: метод навчання – метод зворотного поширення похибки, тип активаційної функції – параметризована випрямлена лінійна функція, кількість прихованих прошарків ШНМ (три) та кількість нейронів у кожному з них (60, 40 та 20 відповідно), функція суматора, кількість вхідних і вихідних нейронів мережі.

В третьому розділі дипломної роботи описано розробку програми для генерації нейромереж вказаної топології, та параметрів. Також, в третьому розділі відбувається підготовка даних та тренування нейромережі. За результатами тестування створеної нейромережі було зроблено висновки, що нейромережевий додаток має достатню

точність для визначення параметрів автоматичного регулятора. Проте, є можливість підвищити точність при використанні більшої кількості нейронів у прихованих шарах та біль різноманітної навчальної вибірки.

Також, у третьому розділі за допомогою створеної програми згенеровано нейромережу для вирішення другої задачі – визначення номера регулятора з заданого набору регуляторів, який доцільно використовувати при певному відхиленні динамічної системи для повернення її до стану рівноваги з найкращим показником якості перехідного процесу. Для цього згенеровано та приведено до формату прийнятного для навчання ШНМ набір навчальних даних. На вхід даної нейромережі подаються координати точки початкового відхилення на фазовій площині, а на виходах отримуємо сигнали, які інтерпретуємо як рівні доцільності використання кожного регулятора для даного відхилення. Створена ШНМ інтегрована у додаток, який на основі результатів обчислення нейромережею координат видає номер регулятора, який необхідно використовувати.

Програма для генерації нейромереж також використовує метод для видалення зв’язків та нейронів, що не використовуються для розрахунків результатів ШНМ.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ 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 с.

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 с.

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, 357с.

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 с. – *Access mode: http://pzs.dstu.dp.ua/DataMining/bibl/practical/Neural%20Network%20Programming%20with%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*

Додаток А

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

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