Міністерство освіти і науки України
Дніпропетровський національний університет
факультет прикладної математики
кафедра математичного забезпечення ЕОМ

МАГІСТЕРСЬКА РОБОТА

Тема: **Методи адаптивного проектування нейронних мереж прямого розповсюдження з каскадною архітектурою**

Виконавець: студент групи ПЗ-09-м Додатко О.В

…….(підпис)

Допускається до захисту

Завідувач кафедри МЗ ЕОМ
Професор Байбуз О.Г.

“….”…….

…….(підпис)

Керівник: доцент кафедри МЗ ЕОМ Кузнєцов К.А

…….. (підпис)

Рецензент: Доц. к-ф. – м.н, ПОМ Громов В.О.

…….(підпис)

Дніпропетровськ
2010 р

РЕФЕРАТ

Магістерська робота: “ Методи адаптивного проектування нейронних мереж прямого розповсюдження з каскадною архітектурою”, с. 59, 11 джерел, 58 рисунків, 8 таблиці, 3 додатки.

Об’єктом дослідження є методи автоматичного підбору архітектури нейронних мереж.

Мета роботи: дослідити властивості існуючих алгоритмів побудови та навчання нейронних мереж, розробити схему кодування архітектури мережі, придатну для застосування у еволюційних алгоритмах. Знайти або створити алгоритм навчання мереж з довільною ациклічною архітектурою. Розробити програмний засіб для проведення експериментів та дослідження запропонованого методу, дослідити його ефективність (порівняно із fnn мережами) на різних задачах.

В процесі роботи було розроблено схему кодування графів на основі G0L граматики, оператори мутації для множини архітектур нейронних мереж. Застосовано алгоритм випадкового пошуку для навчання таких мереж, розроблено програмну реалізацію еволюційного алгоритму автоматичного підбору архітектури, а також проведено її тестування та налагодження. Було досліджено ефективність запропонованого метода на задачі «Two spiral problem» та реальній задачі з металургійної галузі.

В результаті роботи отримано новий спосіб кодування архітектури нейронної мережі з довільною ациклічною архітектурою та застосовано алгоритм LMA для їхнього навчання. Також було розроблено інструмент для автоматичного підбору архітектури мережі. Було проведено серію експериментів, напрямлених на порівняння ефективності запропонованого підходу із традиційними методами.

Результати роботи можуть бути використані для більш ефективного вирішення задач за допомогою нейронних мереж та спростити процес дослідження їх архітектури.

Ключові слова: НЕЙРОННІ МЕРЕЖІ, G0L ГРАМАТИКА, L-СИСТЕМИ, АВТОМАТИЧНИЙ ПІДБІР АРХІТЕКТУРИ, ЕВОЛЮЦІЙНІ АЛГОРИТМИ, ВИПАДКОВИЙ ПОШУК, КАСКАДНА КОРЕЛЯЦІЯ.

ABSTRACT

The graduation research of the 5–year student O.Dodatko (DNU, Applied Mathematics Department) is dedicated to the advanced neural network learning algorithms.

It describes some existing techniques of the architecture fitting and tries to apply one of them to a wider range of the networks. Techniques from neural networking, L-systems, optimization algorithms and evolutionary computations are combined in this work.

During the research a new neural network training framework was designed. It can work with the feedforward networks that have some elements of the cascade architecture. Some tools for the neural networks evolving have been developed as well.

We have also researched the efficiency of the developed framework, comparing to the traditional methods of neural networking (FNN networks).

This work be useful for the students and scientists, dealing with the neural networks and may help them automate the architecture designing process.

Bibliography 11, pictures 58, tables 8, supplement.

Зміст

ВСТУП

Із появою потужних обчислювальних машин, що здатні виконувати мільйони операцій в секунду, з‘явилась можливість втілювати в життя складні алгоритми розв’язання тих чи інших задач реального світу. Однак, традиційна обчислювальна техніка не надто добре пристосована до розв’язання таких задач як розпізнавання образів, класифікація, прогнозування, ітд. Тому постійно ведуться розробки альтернативних способів обчислень. Ідеї багатьох із них запозичені у природи : нейронні мережі, генетичні алгоритми, L-системи ітд.

У наш час штучні нейронні мережі є одним із універсальних засобів розв’язання вказаних задач. Однією з найскладніших проблем при застосуванні такого підходу є вибір правильної архітектури мережі – кількості нейронів, їхніх активаційних функцій, наявності зв’язків між ними.

Побудова архітектури мережі вручну є прийнятною для невеликих за обсягом задач або для задач із вже відомим розв’язком. Але цей підхід не є задовільним при побудові великих мереж для розв’язання складних завдань з реального світу. Тому ведуться розробки методів автоматичного підбору архітектури нейронних мереж, що підходила б якнайкраще до конкретної задачі.

У даній роботі розглянуто деякі існуючі методи навчання нейронних мереж із автоматичним проектуванням її архітектури. Також було здійснено спробу створити програмне забезпечення, здатне знаходити оптимальну для конкретної задачі архітектуру нейронної мережі прямого розповсюдження з каскадними елементами. Це стало можливим в результаті розробки компактної схеми кодування архітектури нейронної мережі та її поєднання із еволюційними алгоритмами.

Отриману систему було перевірено на типовій задачі «Two spirals problem» та реальній задачі з металургійної галузі. Також у роботі наведено порівняння ефективності запропонованого підходу із традиційним підходом, що ґрунтується на використанні FNN мереж.

ПОСТАНОВКА ЗАДАЧІ

Нехай маємо набір вхідних ***X*** та бажаних вихідних ***Y*** даних для нейронної мережі (***X***, ***Y*** – двовимірні масиви даних).

Треба знайти таку архітектуру мережі -- правила її побудови, або безпосередньо її топологію, яка найкраще підходить для розв'язання даної задачі, тобто, навчання завершується із мінімальною помилкою.

Описана задача складається з таких кроків :

1. Розробити компактну схему кодування архітектури нейронної мережі.
2. Розробити засоби перетворення між традиційною схемою кодування (матриця суміжності вершин графа мережі) та розробленою схемою.
3. Розробити оператори мутації для архітектури нейронної мережі в запропонованій схемі кодування.
4. Знайти алгоритм, придатний до навчання мереж прямого розповсюдження із елементами каскадної архітектури
5. Провести тестування обраного алгоритму навчання на задачі „Two Spirals Problem".
6. Застосувати генетичний алгоритм до процесу пошуку оптимальної архітектури (використавши розроблені раніше схему кодування, алгоритм навчання та оператори мутації).
7. Провести тестування розробленого програмного засобу на задачі з металургійної галузі.
8. Порівняти ефективність запропонованого підходу із ефективністю FNN мереж.

ОСНОВНА ЧАСТИНА РОБОТИ

1. Алгоритм каскадної кореляції

Алгоритм каскадної кореляції є алгоритмом навчання та побудови архітектури штучних нейронних мереж. Замість простого коректування ваг в мережі фіксованої топології, каскадна кореляція починається з мінімальною мережі, а потім при навчанні автоматично додає нові приховані нейрони один за іншим, створюючи багатошарову структуру.

Після того, як новий прихований блок був включений в мережу, його вхідні ваги заморожуються. Потім він стає постійним детектором деякої ознаки в мережі.

Каскадна архітектура має ряд переваг у порівнянні з існуючими алгоритмами:

1. Нейронна мережа на її основі швидко навчається
2. Мережа автоматично визначає свої власні розміри і топологію
3. Мережа зберігає засвоєну інформацію за будь-яких змін вхідних даних при повторному навчанні.
4. Навчання не потребує зворотного розповсюдження сигналів помилки (backpropagation).

1.1 Проблеми алгоритму зворотного розповсюдження помилки

Алгоритм каскадної кореляції був розроблений С.Фальманом [2] у спробі подолати деякі недоліки і обмеження популярних алгоритмів зворотного розповсюдження (або "backprop"). Найбільш важливим із недоліків є повільний процес навчання мережі. Навіть на простих базових задачах, що зазвичай використовуються для порівняння алгоритмів навчання, алгоритм зворотного розповсюдження (backpropagation) може потребувати багато тисяч епох навчання, щоб навчитися з прикладів бажаної поведінки. (Епоха визначається як один прохід через весь набір навчальних прикладів.)

С.Фальман [2] визначив дві основні проблеми, які вносять вклад у повільність :

1. проблема розміру кроку
2. проблема руху мішені.

1.1.1 Проблема розміру кроку

Ця проблема виникає тому, що стандартні методи зворотного розповсюдження помилки обчислює тільки $\frac{\partial E}{\partial w}$ – часткову першу похідну від загальної функції помилки, що враховує усі ваги в мережі. Маючи ці

похідні, ми можемо використати метод градієнтного спуску в просторі ваг, зменшуючи помилку при кожному кроці. Легко показати, що якщо рухатись (змінювати ваги) вздовж градієнту нескінченно малими кроками, то досягнемо точки локального мінімуму функції помилки. Досвід показав, що в більшості випадків у точці цього локального мінімуму буде досягатися глобальний мінімум (тобто, шуканий розв’язок), або "достатньо хороший розв’язок".

На практиці, для збільшення швидкості навчання, намагаються обрати якомога більший допустимий крок руху вздовж градієнта. Якщо обрати занадто великий крок, то не буде надійно досягатися хороше рішення. Для того щоб вибрати прийнятний розмір кроку, ми повинні знати не тільки кут нахилу функції помилки, але дещо про її похідні вищих порядків (наприклад, її кривизну в околі поточної точки в просторі ваг). Ця інформація не використовується у стандартному алгоритмі зворотного розповсюдження помилки.

Для розв’язаня цієї проблеми було створено багато методів :

- Метод спряжених градієнтів
- Методи динамічної зміни кроку у нових епохах, враховуючи зміни градієнту.
- Апроксимація функції другої похідної та її використання для обчислення довжини кроку.
- Quickprop (описаний у [2])

1.1.2 Проблема руху мішені

Проблема полягає в тому, що кожен нейрон у прихованих шарах мережі намагається перетворитися на детектора деякої ознаки, який зробить свій корисний внесок у загальну роботу мережі. Але його задача значно ускладнюється тим фактом, що всі інші елементи змінюються одночасно із ним у процесі навчання. Нейрони прихованого шару не можуть взаємодіяти один із одним безпосередньо. Кожен блок «бачить» тільки свій внесок, а також сигнал помилки, який передається з виходів мережі. Сигнал помилки визначає завдання, які окремий нейрон намагається вирішити, але ці завдання постійно змінюється. Замість того, щоб кожен нейрон рухається до цілі швидко і брав безпосередньо на себе деяку корисну роль, ми бачимо складний танець усіх нейронів, що забирає багато часу для врегулювання [2].

Багато дослідників повідомили, що навчання методом зворотного розповсюдження різко сповільнюється (можливо, навіть експоненціально) при збільшенні кількості прихованих шарів у мережі. Зокрема, це уповільнення спричинене ослабленням та розмиванням сигналу помилки при його поширенні по мережі.

Один із розповсюджених проявів проблеми рухомих цілей проблемою є те, що Фальман [2] називає ефектом стада. Припустимо, ми маємо дві обчислювальних підзадачі,– **A** і **B** ,– які повинні бути розв’язані прихованими шарами мережі. Також у нас є множина прихованих нейронів, кожен з яких можна було б використати у будь-якій з цих двох задач. Оскільки нейрони не можуть взаємодіяти один з одним, кожен із них має самостійно вирішувати, яку з цих двох проблем він буде вирішувати. Якщо задача **A** породжує велику кількість або більш когерентні сигнали помилок, ніж завдання **B**, є тенденція для всіх нейронів зосередитися на сигналі **A** та ігнорувати сигнал **B**. Після того, як завдання **A** вирішено, вони можуть зайнятися розв’язанням завдання **B**, яке лишилося єдиним джерелом помилок. Однак, якщо всі обчислювальні одиниці починають рухатися в напрямку **B,** одразу з'являються помилки у розв’язку підзадачі **A**. У більшості випадків, "стадо" з елементів буде розділене на дві частини (та, що розв’язує завдання **A** та завдання **B** відповідно) і розглядати обидва підпункти завдання одночасно. Але може існувати тривалий період нерішучості, перш ніж це станеться. Початкові значення ваг з’єднань в мережі зворотного розповсюдження обирається випадково для запобігання однаковій поведінці всіх нейронів. Але ця початкова мінливість, як правило, зникає у процесі навчання мережі.

Одним із способів боротьби з проблемою рухомих цілей полягає в тому, щоб дозволити лише деяким із нейронів мережі змінюватись одночасно. При цьому, інші нейрони повинні залишатися незмінними. Алгоритм каскадної кореляції використовує цей принцип, дозволяючи лише одному нейрону змінюватися в будь-який даний момент часу.

1.2 Опис алгоритму

Алгоритм каскадної кореляції поєднує в собі дві ключові ідеї:

1. Каскадна архітектура.

   Вона передбачає, що приховані нейрони додаються до мережі по одному і не змінюються після того, як вони були додані.

2. Алгоритм навчання.

   Він створює і додає до мережі нові приховані нейрони. Кожний новий прихований блок додається таким чином, щоб збільшити ***кореляцію*** між новим нейроном та сигналом залишкової помилки, яку треба усунути.

Побудова каскадної мережі починається із декількох вхідних та вихідних нейронів. Їхня кількість однозначно визначається умовами задачі. Кожен із вхідних нейронів приєднано до кожного з вихідних нейронів. Кожне з таких з'єднань має вагу, що може бути налаштована у процесі навчання.

Таким чином, проводиться навчання мережі, що не містить прихованих нейронів (Рис. 1). Прямі з'єднання вхідних та вихідних нейронів навчаються якнайкраще з використанням усіх прикладів.

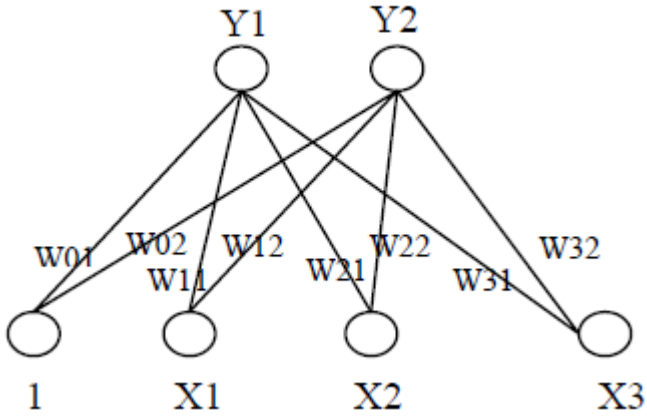

Рис. 1 : Початкова мережа каскадної архітектури з трьома вхідними та двома вихідними нейронами

Оскільки мережа не містить прихованих шарів, які треба навчати, то немає необхідності у використанні зворотного розповсюдження помилки (backpropagation). Натомість, можна застосувати правила Відроу-Гоффа (Widrow-Goff rule), "Дельта" (delta rule), алгоритм навчання перцептрона (Perceptron learning algorithm), quickprop або будь-який з інших відомих алгоритмів навчання для одного шару мереж.

Критерієм зупинки цього етапу навчання є відсутність суттєвих змін величини помилки протягом декількох циклів навчання (кількість цих циклів називається „**параметром терпіння**").

Після цього здійснюється останній прохід по всій навчальній вибірці для вимірювання похибки. Якщо якість навчання мережі задовільна, то алгоритм завершує свою роботу. Інакше наявна залишкова помилка, яка буде зменшуватись у подальшому навчанні. Подальше навчання мережі здійснюється за допомогою додавання прихованих нейронів.

Додавання нейронів відбувається у два етапи :

1. Нейрон-кандидат під'єднується до вхідних нейронів та до усіх прихованих нейронів, які було додано раніше. З вихідними нейронами він поки що не з'єднується (Рис. 2).

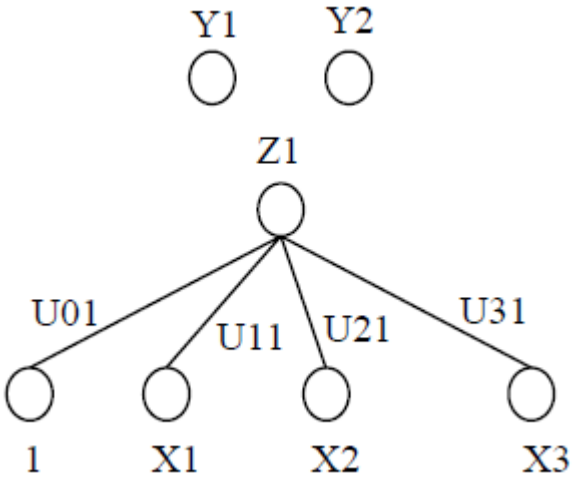

Рис. 2 : Додавання нейронів до каскадної мережі

Ваги вхідних з'єднань цього нейрона змінюють таким чином, щоб отримати якомога більшу кореляцію між значеннями на виході нейрона-кандидата та залишковою помилкою основної мережі. Кореляція обчислюється по такій формулі:

$$S = \sum_{o}^{outputs} \left| \sum_{p}^{examples} (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right| \qquad (1)$$

Позначення :

- o – поточний вихідний нейрон, для якого обчислюється помилка.
- p – поточний приклад з навчальної вибірки.
- V_p – значення на виході нейрона-кандидата для даного прикладу **p**
- $\bar{V}$ – середнє значення на виході нейрона-кандидата по всій навчальній вибірці

- $E_{p,o}$ – значення помилки на виході **o** для навчальної вибірки **p**
- $\bar{E}_o$ – середнє значення помилки на виході **o** по всій навчальній вибірці

Для максимізації цієї функції обчислюють

$$\frac{\partial S}{\partial w_i} = \sum_{o}^{outputs} \sum_{p}^{examples} \sigma_o (E_{p,o} - \bar{E}_o) f_p^{(1)} I_{i,p} \qquad (2)$$

Позначення :

- o – поточний вихідний нейрон, для якого обчислюється помилка.
- p – поточний приклад з навчальної вибірки.
- $E_{p,o}$ – значення помилки на виході **o** для навчальної вибірки **p**
- $\bar{E}_o$ – середнє значення помилки на виході **o** по всій навчальній вибірці
- $f_p^{(1)}$ – перша похідна функції активації нейрона-кандидата, обчислена при роботі мережі з навчальним прикладом **p**
- σ_o – знак кореляції між значенням на виході нейрона-кандидата і вартості виробництва і вихідним нейроном **o**.

$$\sigma_o = sign\left[\sum_{p}^{examples} (V_p - \bar{V})(E_{p,o} - \bar{E}_o) \right] \qquad (3)$$

- $I_{i,p}$ – значення, що нейрон-кандидат отримує на вході від нейрона **i** при роботі мережі з навчальним прикладом **p**

Для покращення роботи алгоритму можливе навчання декількох нейронів-кандидатів одночасно. Після їхнього тренування до мережі вводиться той кандидат, для якого більше значення **S**. Дозволяється мати нейрони-кандидати з різними функціями активації, як-то :

- Лінійні
- Сігмоїдальні

- Гауссівські
- Радиальні базисні

2. Обраний нейрон-кандидат підключається до мережі. Він з'єднується з виходами мережі. Після цього відбувається підбір ваг його вихідних з'єднань.
   Для цього можна використати один із алгоритмів навчання для одного шару мереж (див. вище ст.10). У процесі навчання нейрона-кандидата ваги інших з'єднань залишаються незмінними.

Процес додавання прихованих нейронів повторюється допоки не отримаємо задовільні результати роботи мережі (помилку на виході) або не буде перевищена максимальна кількість епох навчання.

Нами було досліджено реалізацію алгоритму каскадної кореляції "***Freeware academic implementation of the Cascade Correlation Algorithm for Matlab and Octave***", зроблену ***Silva, J.D.S*** [3]. Цю реалізацію було дещо модифіковано для більш зручного її налаштування та пришвидшення роботи за допомогою можливості відключення деяких непотрібних логів. Її поведінку було досліджено на задачі XOR.

При використанні навчальної вибірки

```
p = [ 0 0; 0 1; 1 0; 1 1]';t = [ 0; 1; 1; 0]';
```

Було отримано такі результати :

```
# of added neurons 6;
  Epoch 11000;
 SSE/Current error: 0.020000/1.999714;
Least error achieved: 1.999714
SSE achieved so far: 0.019831

ans = -1.8243   -1.9120    0.0531    1.0632    1.4420   -1.0873   -0.6413
p =
         0         0    1.0000    1.0000
         0    1.0000         0    1.0000
    0.0173    0.0000    0.0000    0.0000
    0.9854    1.0000    1.0000    1.0000
    0.4720    0.0160    0.0162    0.0003
    0.2800    0.9564    0.9668    0.9984
    0.4243    0.9643    0.9641    0.9957
    0.5112    0.0268    0.0289    0.0041

out =    0.0595    0.9373    0.9330    0.0742
```

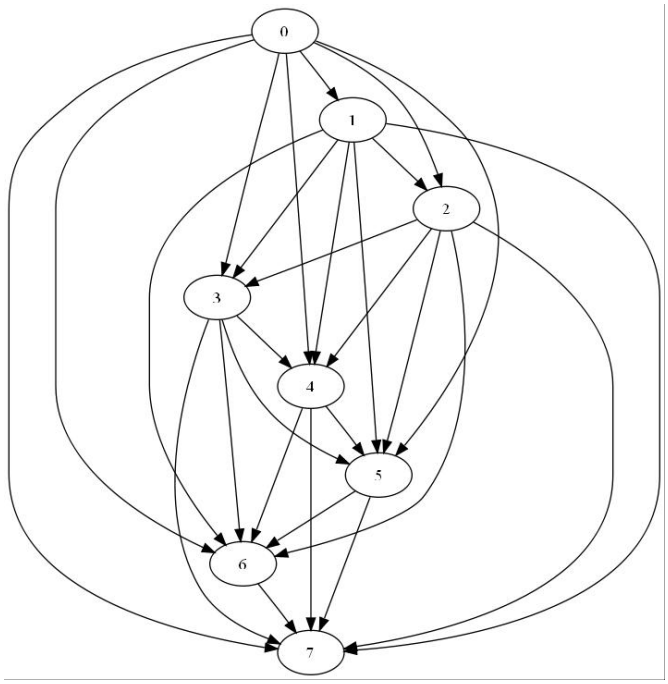

Рис. 3 : Каскадна мережа для розв’язання задачі XOR

Отже, задачу було розв’язано за допомогою мережі, що містить 6 прихованих нейронів (Рис. 3). При цьому було досягнено прийнятної точності 0.019831.

2. Граматика G0L

При зростанні потужності комп'ютерної техніки, стає можливою побудова більш великих штучних нейронних мереж. Обмеження на розмір мережі будуть поступово зникати, і буде дедалі важче їх розробляти та розуміти принципи їхньої роботи. Ось чому дуже важливо знайти добре масштабовані методи для побудови мереж великих розмірів.

При розробці штучних нейронних мереж, часто виникає питання про те, як вибрати початкову архітектуру мережі.

Архітектура мережі складається із

- Топології мережі.
  Вона визначає кількість шарів, кількість нейронів у шарі, та спосіб з'єднання нейронів.
- Активаційних функцій нейронів

Вибір спеціальної архітектури визначає в значній мірі, які функції можуть бути реалізовані в нейронної мережі. Архітектура мережі також має сильний вплив на здатність алгоритму навчатися відтворювати цільову функцію. Ця здатність, звичайно ж, залежить від самої задачі. У загальному випадку, початкова архітектура мережі визначає загальну поведінку системи. Наприклад: мережі, яка призначена для якомога ближчою апроксимації функції та мережі, яка повинна узагальнити як можна більше, знадобляться різні архітектури. Вибір правильної початкової архітектури є однією з основних проблем, що виникають при застосуванні нейронних мереж для складних завдань. Має бути очевидно, що практично неможливо вручну розробляти архітектуру мережі для великих проблем, оскільки це досить складно зробити навіть для порівняно невеликих задач.

Існує дуже мало теоретичних знань про те, як знайти підходящу архітектуру мережі для конкретної задачі. Відсутність цих знань робить практично неможливим знаходження найкращої архітектури мережі вручну або за допомогою деякого аналітичного методу. Навіть для найпростіших мереж, дуже складно виконати повний аналіз їх роботи.

Еволюційні обчислення є одним з небагатьох методів оптимізації, який можна застосувати за відсутності аналітичних знань про задачу. Таким чином, цей метод може бути використаний для автоматичного пошуку підходящих архітектур мережі. У еволюційних обчисленнях популяція розв'язків-кандидатів, або індивідів, обробляється й оптимізується за допомогою відбору та розмноження. При оптимізації архітектури нейронної мережі, кожний індивід містить її закодований опис. Також потрібні правила оцінки . Швидкість та якість роботи мережі потім перетворюють у міру якості (fstness measure), яку еволюційний алгоритм використовує для визначення ймовірності вибору цього індивіда.

Ключовою концепцією в дослідженні штучних нейронних мереж є модульність. Повністю з'єднані штучні нейронні мережі, як правило, працюють гірше, ніж мережі, які не в повній мірі з'єднані, але мають модульну архітектуру, специфічну для конкретної задачі. Як правило, якщо проблема може бути розділена на декілька підзадач, розділення обов'язків між підмережами дозволяє швидше знайти вирішення проблеми. Крім того, що розробка модульних мережевих архітектур для конкретної задачі приведе до поліпшення характеристик, модульність також забезпечує більш масштабовані схеми кодування. При розробці великих мереж із допомогою еволюційних обчислень треба знайти масштабовану систему кодування, що переводить представлення мережі-кандидата у код в хромосоми індивіда.

Масштабованість означає, що обсяг кодування для представлення мережі не зростає занадто сильно при збільшенні розміру мережі. Дуже важливо, зменшити обсяг необхідного коду для зменшення простору пошуку еволюційного алгоритму. Необхідність у розробці великих штучних нейронних мереж дало поштовх для робіт з пошуку схем кодування, які використовують граматики графів, які масштабуються і можуть легко генерувати модульні мережі. Ідея таких схем кодування полягає в тому, щоб кодувати рецепт (тобто, послідовність дій, закони росту мережі) побудови мережі замість повного кодування архітектури.

Для багатьох завдань мережа не може бути навчена легко при використанні алгоритмів навчання звичайних повністю з'єднаних багатошарових нейронних мереж Ось деякі з можливих причин :

- Мережа є занадто малою і не може реалізувати правильну поведінку. Має бути очевидно, що мережа не зможе навчитися правильній поведінці. Зауважимо, можливість реалізації не завжди означає можливість навчання
- Тільки навчальні приклади засвоюються правильно. Інші приклади, які не були розглянуті при навчанні, дають невірні результати. Кажуть, що така мережа має погані узагальнюючі властивості.
- Іноді можливості мережі з узагальнення є добрими після короткого періоду навчання, але зменшується при подальшому навчанні. Мережа стає перенасиченою інформацією (**overtrained**) Це може статися, якщо в мережі занадто багато ступенів свободи.
- Дві або кілька підзадач, які повинні бути вирішені, змагаються між собою за ресурси мережі. Навчання може бути майже неможливим у цьому випадку. Ця проблема називається інтерференцією (**interference**).
- Мережа повинна бути достатньо гнучкою, щоб мати змогу засвоювати нові дані, але вона також повинна бути достатньо

стабільною, щоб не забути про раніше засвоєні дані. Такого співвідношення дуже складно добитися при використанні більшості алгоритмів навчання. Ця проблема називається дилемою стабільності та пластичності (**stability-plasticity dilemma**).

- Навчання мережі проходить повільно. Це може бути результатом невдало вибраних початкових ваг з'єднань мережі, а також результатом невдало вибраної архітектури.

Egbert J. W. Boers and Ida G. Sprinkhuizen-Kuyper, розглядаючи **задачу XOR** у своїх роботах [1], довели, що проста мережа **зворотного розповсюдження** (Рис. 4) навчається набагато повільніше, ніж мережа, що має додаткові прямі з'єднання вхідних нейронів із вихідним (Рис. 5).

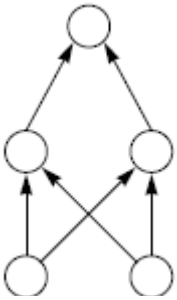

Рис. 4 : класична FNN мережа

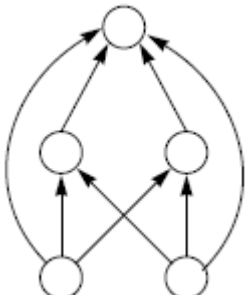

Рис. 5 : Мережа із прямими з'єднаннями

За їхніми результатами, навчання традиційної мережі зайняло **1650 епох**. Навчання ж мережі із додатковими з'єднаннями зайняло лише **30 епох**. При цьому обидві мережі мали однакову величину помилки на виході.

У деяких випадках дуже важливо мати у мережі такі прямі зв'язки між вхідними і вихідними шарами. У загальному випадку, можна розглядати зв'язки між будь-якими шарами і модулями мережі. Існує гіпотеза, що такі зв'язки усувають плоскі ділянки в поверхні помилки.

Отже, архітектура штучної нейронної мережі може сильно вплинути на її здатність до навчання. При використанні нейронних мереж дуже важливо знайти архітектуру, яка найкраще підходить для розв'язання конкретної задачі. Для проблем, невеликих за обсягом або із відомими розв'язками, використовуючи різні методи, можна розробити архітектуру мережі вручну. Однак, коли мова йде про складну проблему, що не має аналітичного рішення, зробити це стає практично неможливим.

Існують два основні підходи до автоматизації процесу проектування архітектури штучної нейронної мережі. Перший підхід полягає в тому, щоб почати з деякої архітектури та додавати або видаляти вузли у процесі навчання. Такі методи поділяються на :

- **Конструктивні**. Початкова архітектура мережі є мінімальною. У процесі навчання до неї додаються нові елементи допоки не буде досягнено задовільних характеристик.
  Усі існуючі конструктивні алгоритми виробляють архітектури, що, зважаючи на їх топології, є проблемо незалежними, оскільки вони додають нові елементи заздалегідь заданим способом. Тобто, у залежності від задачі змінюється тільки розмір мережі.
- **Деструктивні**. Початкова архітектура системи є надмірною. У процесі навчання з неї видаляються деякі елементи та з'єднання для покращення властивостей узагальнення. Елементи видаляються допоки мережа не втратить можливість виконувати свої функції.
  Перш ніж можна почати роботу із деструктивним алгоритмом, потрібно знайти початкову архітектуру. У процесі налаштування мережі інформацію від навчальних прикладів буде поширено поміж усіх ваг, в залежності від їх початкових значень (які задаються випадково). Незважаючи на те, що можна буде збільшити характеристики узагальнення за рахунок зменшення складності мережі, оптимальна модульна архітектура, як правило, не буде знайдена.

Оскільки архітектура мережі в значній мірі впливає на її продуктивність, неможливість знайти оптимальну архітектуру являє собою серйозну проблему. Використання як конструктивних, так і деструктивних методів навчання не буде достатньо, щоб вирішувати складні проблеми.

Розрізняють два різних види схеми кодування :

1. **Схема-план**. Це точний опис архітектури мережі.
   Концепція представлення у вигляді плану передбачає взаємно однозначну відповідність між представленням мережі та її записом у вигляді хромосом.
   Найпростішим способом кодування є матриця суміжності графа мережі.
2. **Схема-рецепт** - це тільки опис механізму, який використовується для побудови архітектури.
   При такому підході в хромосоми кодується не повне представлення мережі, а правила росту архітектури мережі. Цей опис, як правило, являється граматикою запису графів, що визначають стани мережі.

Представлення-рецепти є у значній мірі більш масштабованими порівняно із представленнями-планами. При використанні схеми-плану для кодування великих нейронних мереж, необхідні дуже великі

хромосоми. В результаті, простір пошуку значно зростає, і це ускладнить роботу генетичних алгоритмів підбору архітектури.

Граматика G0L – це граматика запису графів, що базується на механізмі L-систем паралельного переписування строк. При використанні цієї схеми у генетичних алгоритмах, кожен індивід популяції містить набір правил побудови мережі.

L- системи – формалізм переписування строк. Граматика L- системи складається з алфавіту, початкового твердження і набору правил переписування. Початкове твердження (аксіома) переписується з використанням правил : кожне з них описує перетворення певного символа в рядок символів. У той час як у більшості інших граматик правила переписування застосовуються послідовно, в рамках L-системи всі правила переписування застосовуються паралельно.

Грамматика L-системи G мови L визначається за формулою: $G = \{\Sigma, \Pi, \alpha\}$ де Σ – алфавіт мови, Π є кінцевою множиною правил переписування (або правил продукцій) і $\alpha \in \Sigma^*$ – це початковий рядок (аксіома). $\Pi = \{\pi_i \mid \pi_i : \Sigma \to \Sigma^*\}$ це набір правил переписування. Кожне правило переписування π_i визначає перетворення унікального символа Σ, в рядок s : $s \in \Sigma^*$. Усі символи, які не фігурують в лівих частинах правил, за замовчуванням, залишаються незмінними.

Тобто, з граматики :

$\Sigma = \{A, B, C\}$

$\Pi = \{A \to BA, B \to CB, C \to AC\}$, and

$\alpha =$ ABC,

ми отримаємо мову

$L = \{ABC, BACBAC, CBBAACCBBAAC, \ldots\}$

У граматиці для побудови графів визначаються такі символи :

- Цілі числа – з'єднання. Позначають відносну адресу вершини, яка буде з'єднана з поточною. Поточна вершина для даного з'єднання – це найближча зліва до даного з'єднання вершина.
- Символи латинського алфавіту – ідентифікатори вершин. Вони не мають обов'язково бути унікальними. Наявність ідентифікатора просто свідчить наявність вершини.
- Квадратні дужки []. Усе, що знаходиться між ними, являється підсистемою, ізольованим графом. Оскільки у роботі ***„Combined Biological Metaphors" Egbert J. W. Boers and Ida G. Sprinkhuizen-Kuyper*** [1] використовували граматику для побудови лише мереж прямого поширення (feedforward network), вони вважали, що для даної підсистеми вхідними є

вершини, які не мають вхідних з’єднань (мають лише вихідні з’єднання) у межах даного під графа. Відповідно, вихідними є вершини, які не мають вихідних з’єднань (мають лише вхідні з’єднання) у межах даного під графа.
Така підсистема розглядається як одна вершина поточної підсистеми.
До вхідних з’єднань підсистеми приєднуються зовнішні з’єднання, напрямлені до підсистеми. Вихідні з’єднання приєднуються до зовнішніх вершин, з якими дана підсистема з’єднана.

Наприклад, рядок A12B2C01D-3 є записом такого графа (Рис. 6). На даному прикладі добре видно, що мається на увазі під відносною адресою вершини. Ці відносні адреси помічені над дугами графа. Додаткової виразності малюнок набуває завдяки збереженню порядку слідування вершин у рядку.

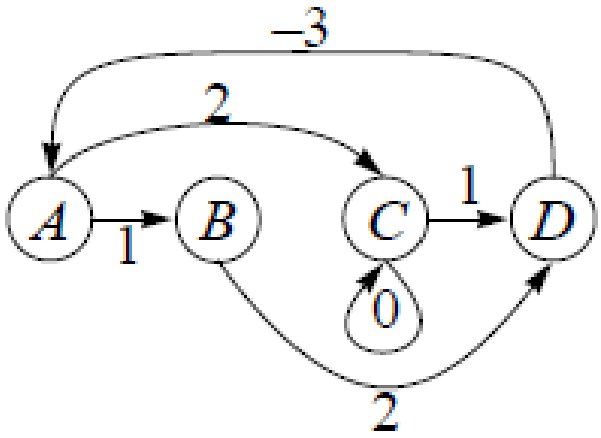

Рис. 6 : Результуючий граф

Наведемо приклад використання модульності. Якщо розглядати L-систему

G = <*N, T, P, S*>

T = *{B;[;];* 1; -2}

N =*{A}*

S = *{A}*

P =*{A* → *[BB]1[AB]1B* – 2}

То після трьох застосувань правил переписування, отримаємо такий граф (Рис. 7) :

[BB]1 [[BB]1 [[BB]1[AB]1B - 2B] 1B - 2B] 1B - 2:

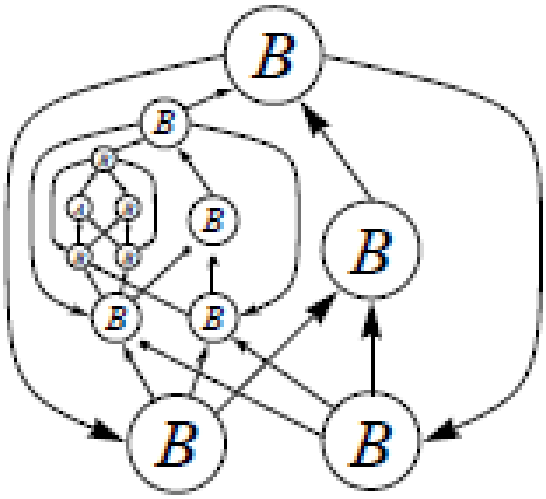

Рис. 7 : Результуючий граф

Легко помітити, що кожен з малих під графів (тих, що розміщуються зліва) є ніби зменшеною копією усього графу (звичайно, якщо розглядати ці підграфи як єдине ціле, тобто, одну вершину). Тобто, даний граф повторює себе, що є характерно для фракталів та L-систем.

3. Модифікація граматики G0L

Проаналізувавши граматику G0L, було знайдено декілька неоднозначностей у ній. Це стосується таких речей як з'єднання компонентів вкладеного графа.

Розглянемо такий граф :
A12[C1D]01-1B

При побудові графа неоднозначним є такі моменти :

- З якими вершинами вкладеного слід з'єднувати вершину A. (З *C, D* або з обома?)
- Які вершини вкладеного графу слід з'єднувати з вершинами A, B. (З *C, D* або з обидві?)
- Які з'єднання слід додати до вкладеного графа, враховуючи наявність петлі?

Визначення строгої та нестрогої модульності не є достатньо чітким для розв'язання цих питань. Навіть якщо вважати вхідними вершини підграфа, до яких нічого не входить (у межах даного підграфа), а вихідними – з яких нічого не виходить, то при розгляді графа *A12[C1D-1]01-1B* підграф *[C1D-1]* не буде мати ані вхідних, ані вихідних вершин.

Для розв'язання цих неоднозначностей було вирішено додати до граматики деякі символи.

Отже, модифікована граматика G0L відрізняється від оригінальної наступним :

- Позначення вершин не обмежені одним символом ім'я вершини може складатися з буквенно-числової послідовності (букви – з латинського алфавіту) та повинне починатися з букви.
- З'єднання не обмежене одною цифрою.
- Необхідно явно задавати знаки для чисел.
- Додано символ „_" (підкреслення) для розмежування імен вершин одне від одного.
- Додано символи для опису атрибутів вершин.

У такому записі рядок символів ***_A:i+1+2[_C:io+1_D:o]+0+1-1_B:o*** буде однозначно описувати граф (Рис. 8) :

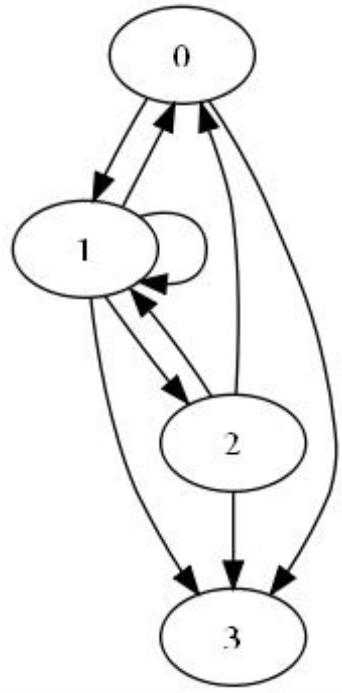

Рис. 8 :
Результуючий граф

Його матриця суміжності :

$$\begin{pmatrix} 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Каскадний граф, який є основою побудови структури мережі для алгоритму каскадної кореляції має такий запис у модифікованій граматиці:

[_AA:io+1[_ABA:io+1[_ABBA:io+1[_ABBBA:io]]]]+1_B:o

У даному прикладі (Рис. 9) наведено граф, отриманий після чотирьох перетворень :

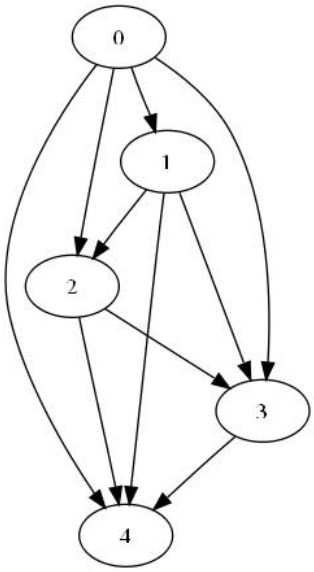

Рис. 9 :
Каскадний граф

Граф, розглянутий у даному прикладі, має таку матрицю суміжності :

$$\begin{pmatrix} 0 & 1 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

У загальному випадку граф описується трикутною матрицею, у якій на головній діагоналі та нижче – нулі, вище – одиниці.

$$\begin{pmatrix} 0 & 1 & 1 & \dots & \dots & \dots & 1 \\ 0 & 0 & 1 & \dots & \dots & \dots & 1 \\ 0 & 0 & 0 & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & \dots & \dots & 0 \end{pmatrix}$$

4. Розбір граматики G0L

4.1 Загальні відомості

Для розбору граматики G0L у модифікованому варіанті було вирішено написати інтерпретатор. Виходячи з особливостей граматики, було вирішено створити інтерпретатор, що працює у два проходи.

За перший прохід відбувається переведення графа, записаного у граматиці G0L у проміжне внутрішнє представлення.

Представлення :

- **Граф** – послідовність **вершин**.
- **Вершина** має такі складові :
  - Ідентифікатор
  - Атрибути
  - Відносні зв’язки з іншими вершинами даного під графа
  - Підграф, що їй відповідає (якщо він порожній, то вершина є простою).

Для програмної реалізації даного етапу було використано програми **Flex** та **Gnu Bison**, які є вільними програмами та розповсюджуються під ліцензією **GPL**. За допомогою цих програм можна отримати код мовою С, який здійснює лексичний та синтаксичний аналіз. Для завдання правил використовується форма запису, близька до загальноприйнятої **РБНФ**.

Обмін даними між інтерпретатором граматики G0L та

4.2 Граматика G0L у формі РБНФ :

Умовні позначення РБНФ.

РБНФ – Розширені Бекусо-Науровських Форми. Для запису граматики використовують такі позначення:

Таблиця 1

Умовні позначення РБНФ

Позначення	Зміст
::=	"за визначенням є"
|	"або"
[X]	0 або 1 раз використання X
{X}	0 або більше разів використання X
(X|Y)	вибір: або X або Y
Xyz	терминальный символ xyz (підкреслений жирний)

Запис граматики G0L :

Graph ::={*Node*}

Node ::=*AbstractVertex* [**:** *Attributes*] [*ConnectionList*]

Attributes ::={*Attribute*}
Attribute ::=**I** | **O**

ConnectionList ::={*Connection*}
Connection = Sign Number
Sign ::=+ | –

AbstractVertex ::= **[***Graph* **]** | ConcreteVertex

Number – десяткова цифра (0..9)

ConcreteVertex – символьний ідентификатор. Допускаются буквы латинського алфавиту, цифри.

4.3 Граматика G0L у записі Gnu Bison

Нижче наведено запис граматики G0L у формі, прийнятній для програми **Gnu Bison**:

```
Graph :
  NodeList
;

NodeList :
  /* empty */
  |
    Node
    NodeList
;


Node :
  AbstractVertex
  OptionalAttributes
  ConnectionList
;

OptionalAttributes :
  /*empty*/
  |
    ATTRIBUTES_HEADER
    ID {SetAttributes(parserInstance);}/*Attributes*/
;

ConnectionList :
  /*empty*/
  |
    CONNECTION {AppendConnection(parserInstance);}
    ConnectionList
;

AbstractVertex :
  Subgraph | ConcreteVertex
;

Subgraph :
  {AppendVertex(parserInstance);}
  LBRACKET {StartSubgraph(parserInstance);}
  Graph
  RBRACKET {EndSubgraph(parserInstance);}
;

ConcreteVertex :
  NODE_SEPARATOR  {AppendVertex(parserInstance);}
  ID {SetVertexName(parserInstance);}
;
```

4.4 Побудова матриці суміжності графа

Після переведення у внутрішнє представлення будується матриця суміжності. Для збереження матриць було вирішено використати матриці з бібліотеки ***boost***.

За поточним підграфом будується матриця суміжності. Після цього виконується розширення вершин.

Якщо вершина є підграфом, то вона видаляється з послідовності та матриці суміжності, а на її місце вставляється підграф. Після вставки виконується уточнення зв'язків та атрибутів її вершин та вершин, що були з нею поєдані.

Вхідні зєднання підєднуються лише до тих вершин підграфа, що мають вхідний атрибут (i).

Лише вихідні зєднання підграфа (ті, що мають атрибут (o)) підєднуються до вершин основного графа.

У разі наявності петлі, ввихідні зєднання підграфа (ті, що мають атрибут (o)) підєднуються до вхідних зєднань підграфа (тих що мають вхідний атрибут (i)).

Проілюструємо роботу алгоритма на прикладі розглянутого вище графа (Рис. 10) ***_A:i+1+2[_C:io+1_D:o]+0+1-1_B***

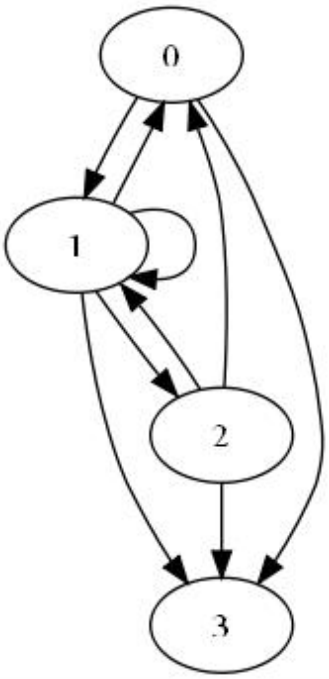

Рис. 10 : Результуючий граф

Expanding matrix :
(0,1,1)
(1,1,1)
(0,0,0)

Inserting matrix :
(0,1)
(0,0)

Inserting to
Row = 1
Col = 1

Expanded matrix (матриця після розширення)
(0,0,0,1)
(0,0,1,0)
(0,0,0,0)
(0,0,0,0)

Updated matrix (матриця після уточнення)
(0,1,0,1)
(1,1,1,1)
(1,1,0,1)
(0,0,0,0)

4.5 Генератор графів

Таким чином, поєднуючи граматику G0L та механізми L-систем переписування символьних строк, було отримано генератор графів. Для реалізації L-систем було використано ***matlab***. Наразі використовуються правила без врахування контексту. При наявності декількох правил, праві частини яких вміщують одна одну, застосовується правило з найдовшою правою частиною. L-системи зберігаються у форматі ***xml*** при використанні ***matlab xml toolbox***. Використовуючи цей генератор, можна простежити еволюцію розвитку архітектури мережі.

ПРИМІТКА : при проектуванні початкових правил слід уникати наявності правил із лівими частинами, одна з яких є підстрокою іншої, оскільки це може призвести до генерації неправильних архітектур.

1. Бінарне дерево.
   G = <N, T, P, S>
   T = {b, +, 1, 2, [,], }
   N = { **a, i, o, :, _**}
   S = {**_a:io** }
   P = {**_a:io → _b:io+1+2[_a:io]:o[_a:io]:o**}

   На цих малюнках представлені дерева після двох, трьох та чотирьох переписувань відповідно.

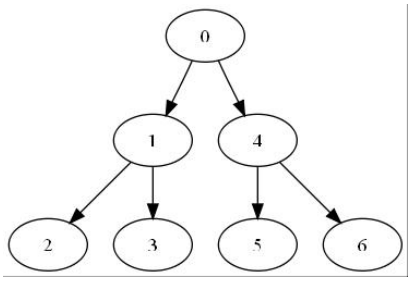

Рис. 11 : Деревовидний граф після двох переписувань

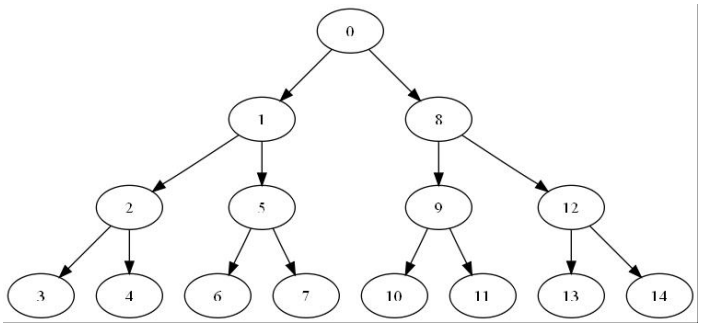

Рис. 12 : Деревовидний граф після трьох переписувань

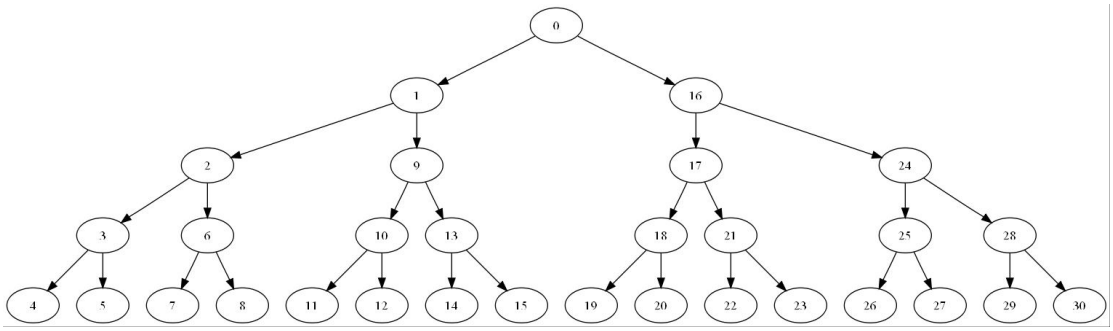

Рис. 13 : Деревовидний граф після чотирьох переписувань

2. Каскадна мережа.

G = <N, T, P, S>
T = {b, +, 1, [,], }
N = { **a, i, o, :, _**}
S = {**[_a:io]:i+1_b:o**}
P = {**_a:io → _b:io+1[_a:io]:io** }

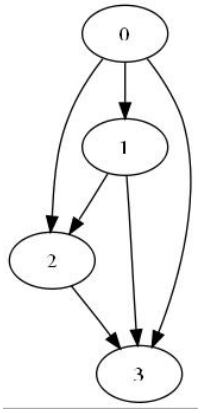

Рис. 14 : Каскадний граф після двох переписувань

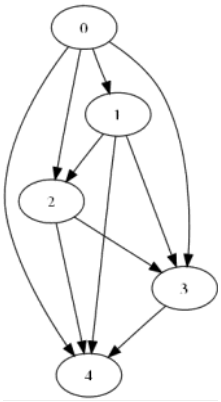

Рис. 15 : Каскадний граф після трьох переписувань

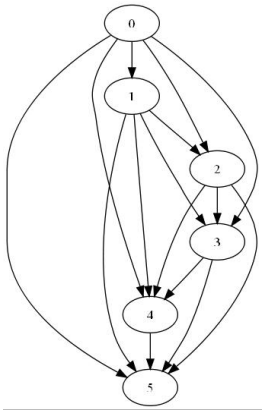

Рис. 16 : Каскадний граф після чотирьох переписувань

3. Expanding cascade

G = <N, T, P, S>

T = {c +, 1, [,] }

N = { **a, d, b, i, o, :, _**}

S = {**[_d:io]:i+1_b:o** }

P = {

_d:io → _c:io+1[_a:io]:io

_b:io → [_d:io]:i+1_b:o

_a:io → _b:io+1[_a:io]:io

}

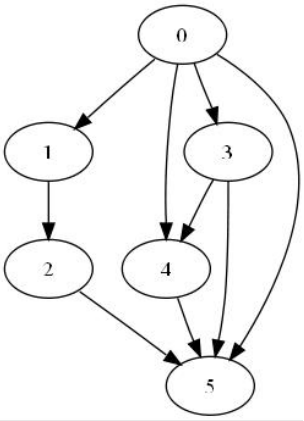

Рис. 18 : Модифікований каскадний граф після двох переписувань

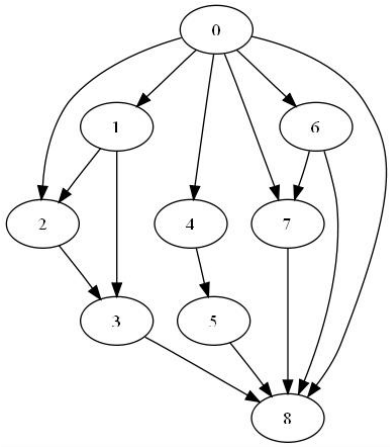

Рис. 17 : Модифікований каскадний граф після трьох переписувань

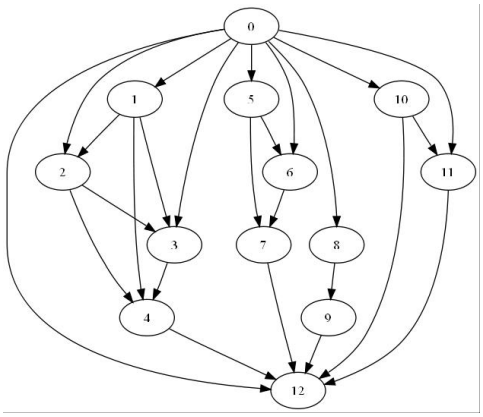

Рис. 19 : Модифікований каскадний граф після чотирьох переписувань

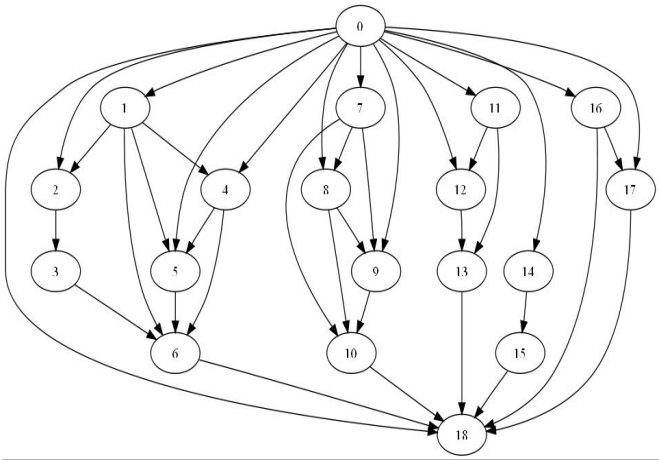

Рис. 20 : Модифікований каскадний граф після п’ятьох переписувань

5. Генератор граматик.

Для визначення граматики із найкращою архітектурою було вирішено використати генетичний алгоритм.

5.1 Генетичниі алгоритми.

Генети́чний алгори́тм (англ. *genetic algorithm*) — це еволюційний алгоритм пошуку, що використовується для розв'язання задач оптимізації і моделювання шляхом послідовного підбору, комбінування і варіації шуканих параметрів із використанням механізмів, що нагадують біологічну еволюцію.

Задача кодується таким чином, щоб її розв'язок був представлений у вигляді масиву подібного до інформації складу хромосоми. Цей масив часто називають саме так «хромосома». Випадковим чином в масиві створюється деяка кількість початкових елементів «особин», або початкова популяція. Індивіди оцінюються із використанням функції пристосування, в результаті якої кожному з них присвоюється певне значення пристосованості. Пристосованість визначає імовірність виживання особи. Після цього з використанням отриманих значень пристосованості вибираються особи допущені до схрещення (*селекція*). До осіб застосовується "генетичні оператори" (в більшості випадків це оператор схрещення (**crossover**) і оператор мутації (**mutation**), створюючи таким чином наступне покоління осіб. Особи наступного покоління також оцінюються застосуванням генетичних операторів і виконується селекція і мутація. Так моделюється еволюційний процес, що продовжується декілька життєвих циклів (*поколінь*), поки не буде виконано критерій зупинки алгоритму. Таким критерієм може бути:

- знаходження глобального, або надоптимального вирішення;
- вичерпання числа поколінь, що відпущені на еволюцію;
- вичерпання часу, відпущеного на еволюцію.

Етапи генетичного алгоритму

1. Створення початкової популяції:
2. Обчислення функції пристосованості для осіб популяції (оцінювання)
3. Повторювання до виконання критерію зупинки алгоритму:
    1. Вибір індивідів із поточної популяції (селекція)
    2. Схрещення або/та мутація
    3. Обчислення функції пристосовуваності для всіх осіб
    4. Формування нового покоління

5.2 Основні позначення

Застосовуючи генетичні алгоритми для розв'язання задачі підбору архітектури мережі, у якості індивідів (особин у популяції) будемо розглядати L-системи, що описують архітектуру мережі. У якості хромосом будемо розглядати правила, що входять до L-системи. У якості міри пристосованості візьмемо значення помилки на виході мережі після процесу навчання.

У наведеній нижче таблиці подано еквівалентні значення термінів, що було використано для опису підбору архітектури мережі та генетичних алгоритмів.

Таблиця 2

Застосування генетичних алгоритмів у задачах підбору архітектури мережі

Генетичні алгоритми	Підбір архітектури мережі
Індивід	L-система
Хромосома	Правило L-системи, Аксіома L-системи
Міра пристосовуваності	Помилка на виході мережі (6510b)
Оператор схрещування (**Cross-over operator**)	Не застосовано (N/A)
Оператор мутації (**Mutation operator**)	• Заміна вершини на шар нейронів • Заміна вершини на дерево • Заміна вершини на каскад

5.3 Операція схрещування

Для спрощення задачі пошуку було вирішено не використовувати оператор схрещування (**cross-over operator**). Використання цього оператора потребує розв'язання декількох складних проблем :

1. Задача перевірки результатів роботи оператора.
   Не всі правила, створені за допомогою цього оператора будуть вірними з точки зору G0L граматики.
2. Проблема об'єднання (**merge**) двох L-систем та збереження їх просторів імен (**namespaces**).
   Нехай маємо два набора правил L-системи :
   Набір 1

```
P1 = {
_a:io  => _b:io+1[_a:io]:io
}
```

Набір 2

```
P2 = {
_a:io       =>      _b:io+1[_c:io]:io
_b:io       =>      _c:io
}
```

Слід розрізняти змінні „a” та „b” з першого набору та змінні „a” та „b” з другого набору. При об’єднанні правил слід роз’вязати задачу перейменування змінних. При цьому залежності між ними, які мали місце у початкових системах, повинні зберігтися у новій системі.

Результатом об’єднання (merge) наведених прикладів повинна стати приблизно така система.

```
P12 = {
_a:io       =>      _b:io+1[_c:io]:io
_b:io       =>      _c:io
_x:io       =>      _y:io+1[_x:io]:io
}
```

5.4 Операція мутації

Отже, для зміни особин популяції було використано лише операцію мутації. Зважаючи на специфічне кодування хромосом, застосування класичних операторів на кшталт інверсії випадкового біту хромосоми не є можливим.

Таким чином, було розроблено деякі власні оператори мутації. Наразі, ці методи зводяться до додавання правил визначеного виду до складу L-системи.

5.4.1 Заміна вершини на шар нейронів

До L-системи вводиться правило виду :

```
P = { _b     =>      [_x1:io_x2:io_xN:io] }
```

Кількість нейронів, на які вершину буде замінено, обирається випадково. Для зменшення зони пошуку ця кількість є обмеженою. Найбільша кількість вершин, що може бути введена до системи за одне застосування мутації, становить **5**.

5.4.2 Заміна вершини на дерево

До L-системи вводиться правило виду :

```
P = {
_b        =>        [_x:io+1+2+3…+N_b:io_b:io_b:io…_b:io]
}
```

кількість нейронів, на які вершину буде замінено, обирається випадково. Для зменшення зони пошуку ця кількість є обмеженою. Найбільша арність дерева **N**, що може бути введене до системи за одне застосування мутації, становить **3**.

5.4.3 Заміна вершини на каскад.

До L-системи вводиться правило виду :

```
P = {
_b          =>        [[_x:io]:i+1_y:o]
_ x:io      =>        _ y:io+1[_x:io]:io
}
```

При використанні усіх правил, вершина для застосування **b** мутації обирається випадково з-поміж вершин, які можуть з’явитися у результаті переписування L-системи.

Наприклад, для системи

```
P = {
_d:io       =>        _c:io+1[_a:io]:io
_b:io       =>        [_d:io]:i+1_b:o
_a:io       =>        _b:io+1[_a:io]:io
}
```

змінна для застосування оператора мутації може обиратися з множини {c, a, d, b}

5.5 Результати роботи операторів

5.5.1 Заміна вершини на шар нейронів

Мережа з одного нейрона :

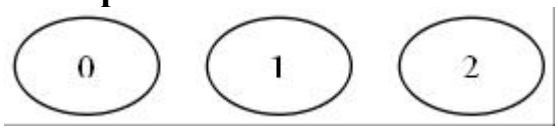

Рис. 21 : Шар нейронів (мутація одиничної вершини)

Аксіома : **A = {_a}**

Правила :

```
P = {
_a          =>          _b
_b          =>          [_x1:io_x2:io_x3:io]
}
```

Каскадна мережа.

Аксіома : **A = { [_a:io]:i+1_b:o }**

Правила :

```
P = {
 _a:io          =>          _b:io+1[_a:io]:io
}
```

Після трьох переписувань :

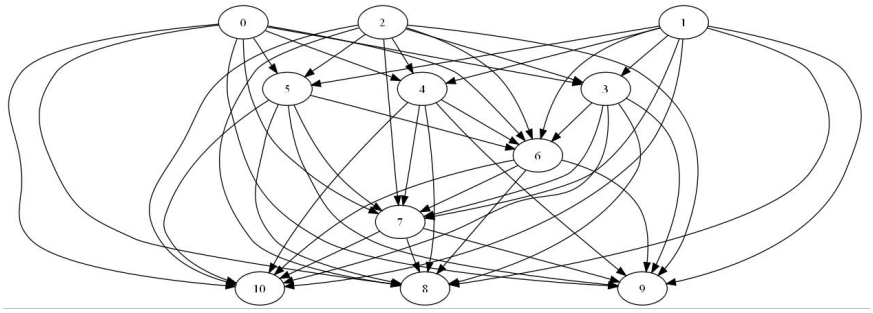

Рис. 22 : Мутована каскадна мережа після трьох переписувань

5.5.2 Заміна вершини на дерево

Мережа з одного нейрона :

Аксіома : **A = { _a }**

Правила :

P = {

_a => _b

_b => [_x:i+1+2_b:o_b:o]

}

Після трьох переписувань :

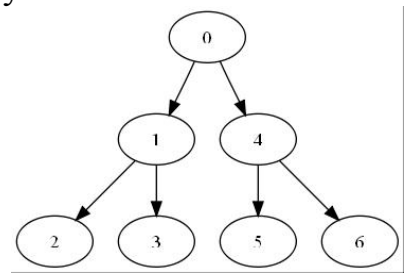

Рис. 23 : Бінарне дерево (мутація одиничної вершини)

Каскадна мережа.

Аксіома : **A = { [_a:io]:i+1_b:o }**

Правила :

P = {

_ a:io => _b:io+1[_a:io]:io

_b => [_x:i+1+2_b:o_b:o]

}

Після трьох переписувань :

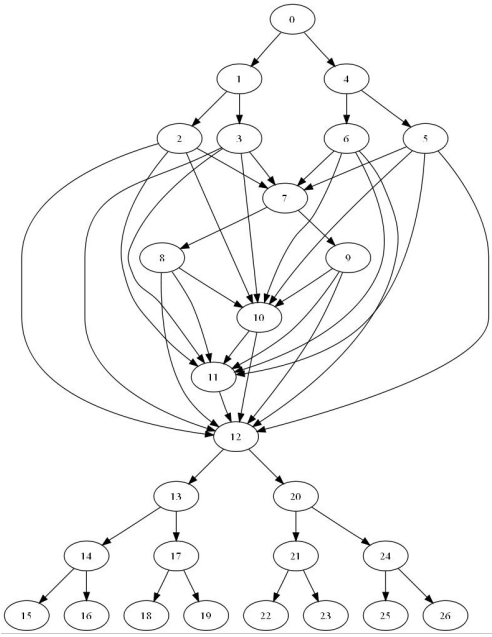

Рис. 24 : Мутована каскадна мережа після трьох переписувань

5.5.3 Заміна вершини на каскад.

Мережа з одного нейрона :

Аксіома : **A = { _a }**

Правила :

```
P = {
_a      =>      _b
_b      =>      [[_x:io]:i+1_y:o]
_x:io     =>      _y:io+1[_x:io]:io
}
```

Після трьох переписувань :

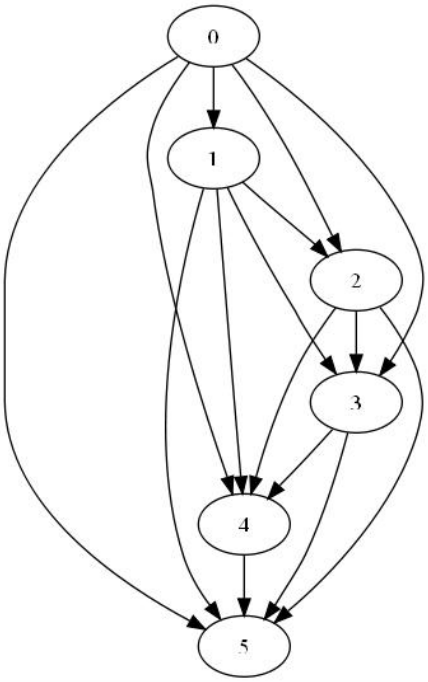

Рис. 25 : Каскад (мутація одиничної вершини)

Каскадна мережа.

Аксіома : **A = { [_a:io]:i+1_b:o }**

Правила :

P = {

_ a:io	=>	**_b:io+1[_a:io]:io**
_b	=>	**[[_x:io]:i+1_y:o]**
_x:io	=>	**_y:io+1[_x:io]:io**

}

Після трьох переписувань:

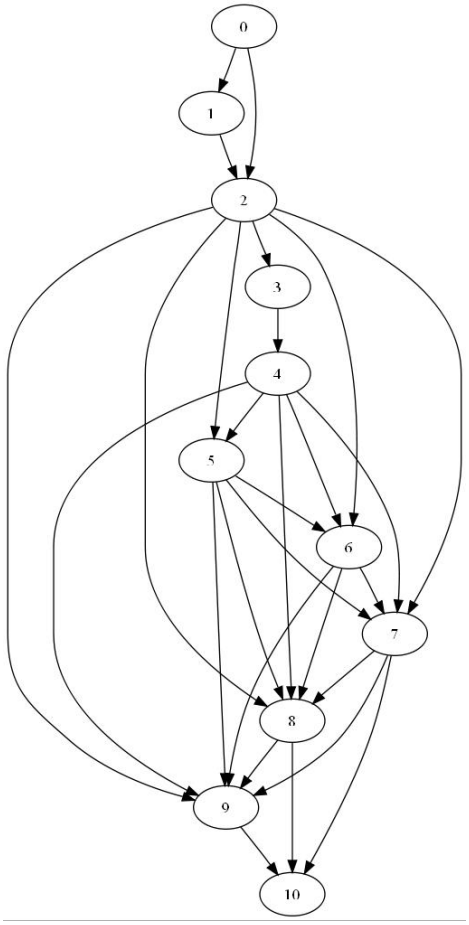

Рис. 26 : Мутована каскадна мережа після трьох переписувань

5.5.3 Додавання шару до мережі

При використанні ізольованого ланцюжка до L-системи вводиться правило вигляду:

Padd = _b →[_b1:i+1_b2:o]

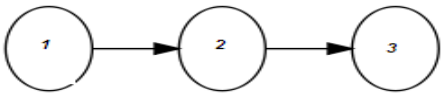

Рис. 27 : до застосування правила

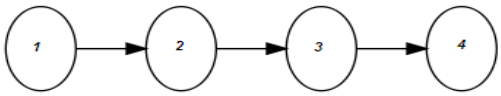

Рис. 28 : після застосування правила

При використанні відкритого ланцюжка до L-системи вводиться правило вигляду:

Padd = _b →[_b1:io+1_b2:io]

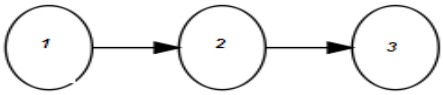

Рис. 29: до застосування правила

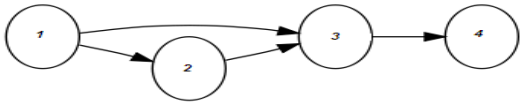

Рис. 30: після застосування правила

6. Алгоритм навчання нейронної мережі

Зважаючи на топологію отриманих нейронних мереж, для їхнього навчання неможливо застосувати стандартні засоби ***Matlab***. Це пов’язано з тим, що топологія не дозволяє виділити шари нейронів та працювати з ними (завдяки додатковим з’єднанням, запозичених з каскадної моделі).

Matlab Neural Network toolbox має можливість роботи з мережами, що мають каскадну архітектуру. Зокрема, там є функція ***newcf*** для створення каскадної мережі. Для навчання таких мереж доцільно використовувати алгоритм LMA (***Levenberg-Marquardt backpropagation algorithm***). Проте, функція ***newcf*** завжди створює повністю зв’язану архітектуру. Тобто, до кожного шару мережі під'єднані всі попередні шари. Отже, використання цієї функції неприйнятно в рамках розглянутої задачі.

Таким чином, виникає потреба самостійно конструювати необхідну архітектуру (***custom network***) у термінах об'єкта ***Matlab Neural Network*** (для більш докладної інформації див [12]). Для цих перетворень було створено ряд функцій-конверторів. При цьому виникли деякі труднощі, пов’язані з особливостями ***Matlab Neural network toolbox*** :

1. Якщо два шари нейронів з'єднані між собою, то налаштовуються всі зв'язки між їх нейронами. Мережі, розглянуті в даній роботі, не потребують такої повної пов'язаності. Навпаки, при їх проектуванні ми намагаємося її уникнути.
2. Неможливо задати властивості (активаційний функцію, "заморожування" ваг і т.д.) для одного конкретного нейрона в шарі. Це можна зробити тільки для шару в цілому. Тому ми пропонуємо використовувати мережу, кожен шар якої складається лише з одного нейрона.
3. Функція ініціалізації шарів "***initnw***" не працює для мережі, що має хоча б один шар нейронів з пороговою (***hardlim***, ***hardlims***) функцією активації. Більш того, навіть мережу з вдалою топологією може виявитися неефективною при поганому виборі початкових ваг зв'язків.
4. Нейрони вхідного шару (***net.inputs***) не описуються в структурах (***net.layers***) і (***net.layerConnect***). Ці нейрони винесені в окремий шар з лінійними активаційними функціями.

Таким чином, при проектуванні в ***Matlab*** мереж із не шаруватою архітектурою, ми пропонуємо використовувати матрицю суміжності графа як структуру ***net.layerConnect***. При цьому необхідно ініціалізувати елементи матриці ***net.IW***, відповідні вхідним нейронам, значеннями **"1"**. Також слід заборонити подальше зміна цієї матриці, задавши ***net.inputWeights (i). Learn = 0*** для індексу і кожного вхідного нейрона.

РЕЗУЛЬТАТИ

Описаний вище алгоритм навчання було випробувано на задачі “Two Spiral Problem” та реальній задачі з області металургії.

Задача „Two Spirals Problem”

Задача полягає у відокремленні червоних та зелених точок площини у дві спіралі. Розглядається обмежена частина площини (Рис. 31).

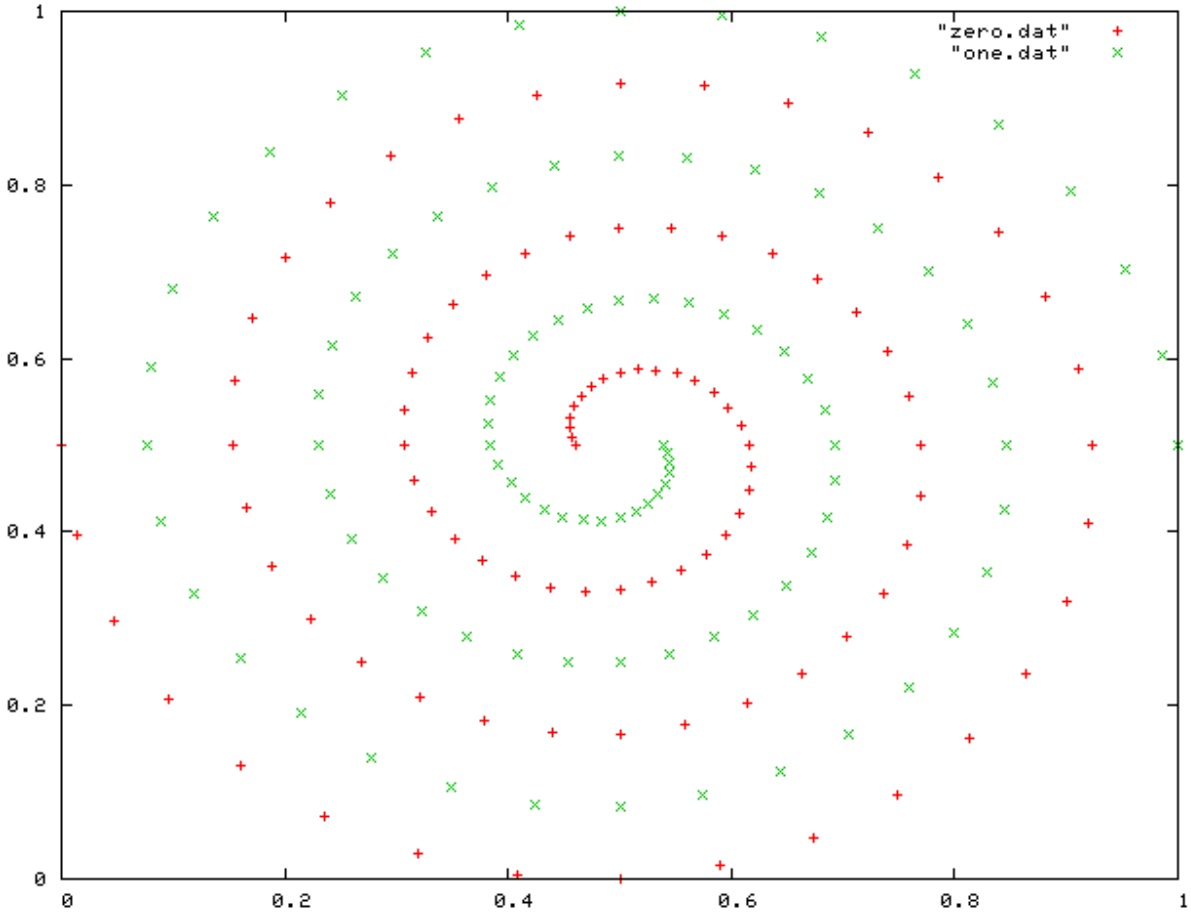

Рис. 31 : Дані для задачі „Two Spiral Problem”

Для навчання було використано навчальну вибірку з 194 точок (див. Додаток 2), зображену на малюнку вище (Рис. 31).

Для розв’язання задачі було побудовано мережу каскадної архітектури з 16 нейронів Кількість нейронів була взята як найбільш вірогідна. Вибір кількості нейронів базувався на результатах роботи Фальмана [2] та результатах навчання мережі з проекту Pyro [10].

Спіралі, зображені на Рис. 31, описуються за допомогою формул (4) та (5):

$$x = r \cdot \cos\alpha, y = r \cdot \sin\alpha \quad (4)$$

$$\alpha = \frac{k \cdot \pi}{16 \cdot d}, r = \frac{R \cdot (104 \cdot d - k)}{104 \cdot d}, k \in [0;\ 96 \cdot d], k \epsilon Z \quad (5)$$

Випробування проводилися як на архітектурах, заданих вручну, так і на архітектурах, отриманих в результаті автоматичного підбору. Для вирішення цього завдання [13] була запропонована топологія 2-2-8-1. При цьому кожен шар з'єднаний з усіма наступними шарами (Рис. 32).

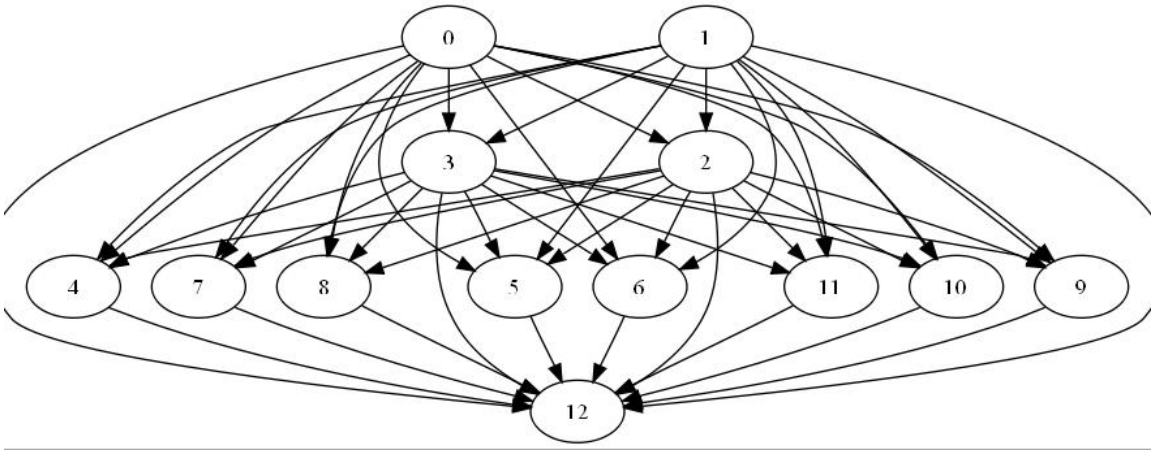

Рис. 32 : архітектура 2-2-8-1

На Рис. 33 представлені результати навчання такої мережі.

Рис. 33 : результати навчання мережі 2-2-8-1

На Рис. 34 приведені результати навчання каскадної мережі, що складається з 16 шарів.

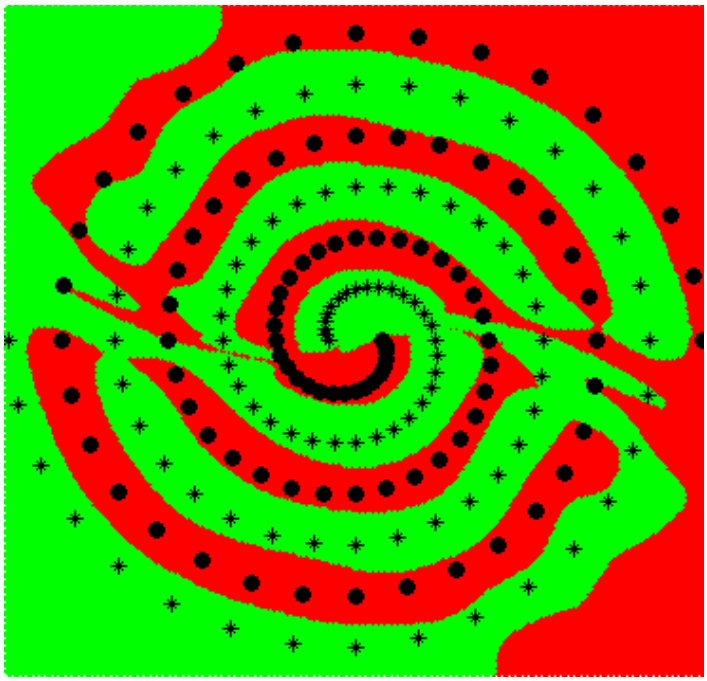

Рис. 34 : каскадна мережа з 16 шарів

Після отримання задовільних результатів навчання для статичних архітектур було проведено декілька експериментів з автоматичного підбору. На рис. 35 наведено результати найбільш вдалого з них.

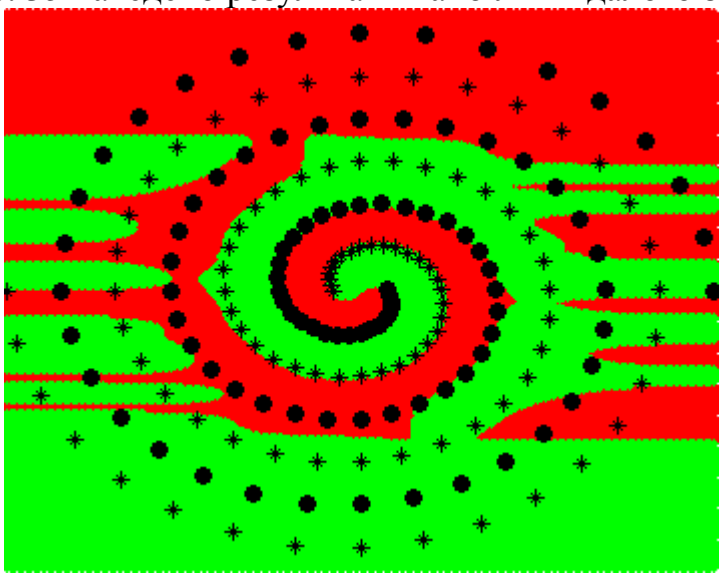

Рис. 35 : результат автоматичного підбору

На Рис. 36 – Рис. 41 можна простежити побудову топології результуючої мережі.

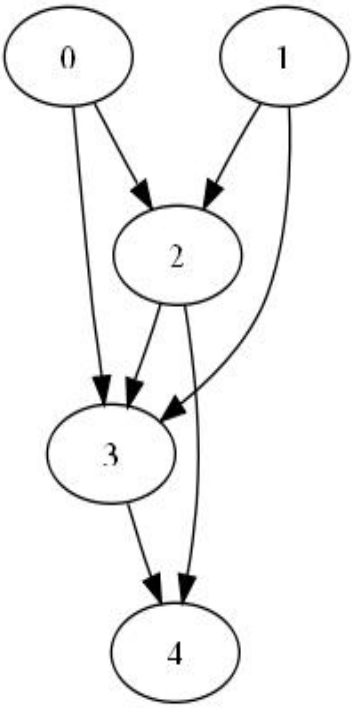

Рис. 36 : еволюція 2 кроки

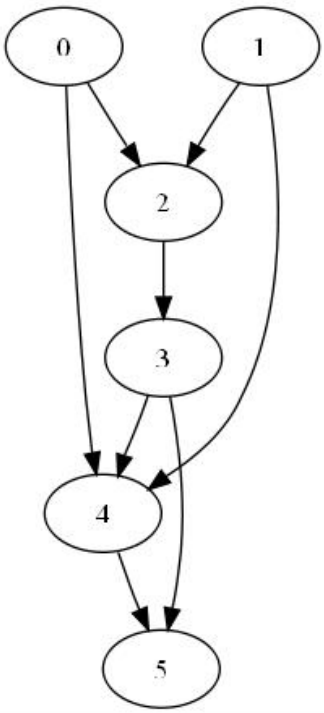

Рис. 37 : еволюція 3 кроки

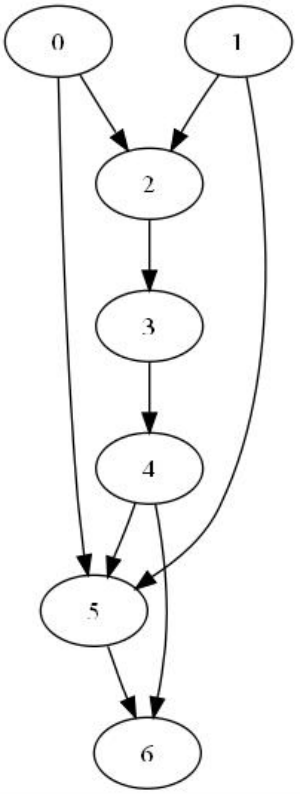

Рис. 38 : еволюція 4 кроки

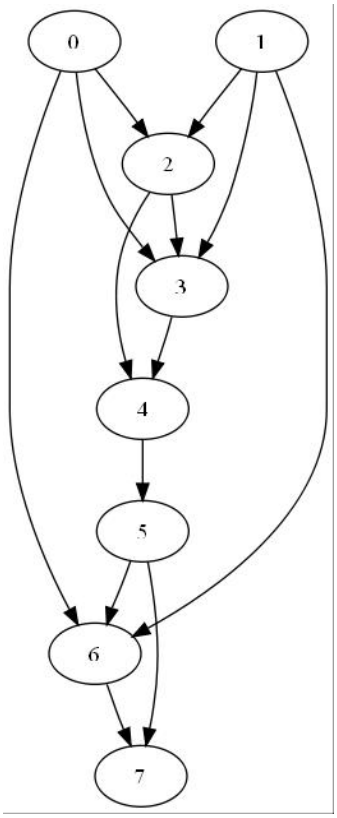

Рис. 39 : еволюція 5 кроів

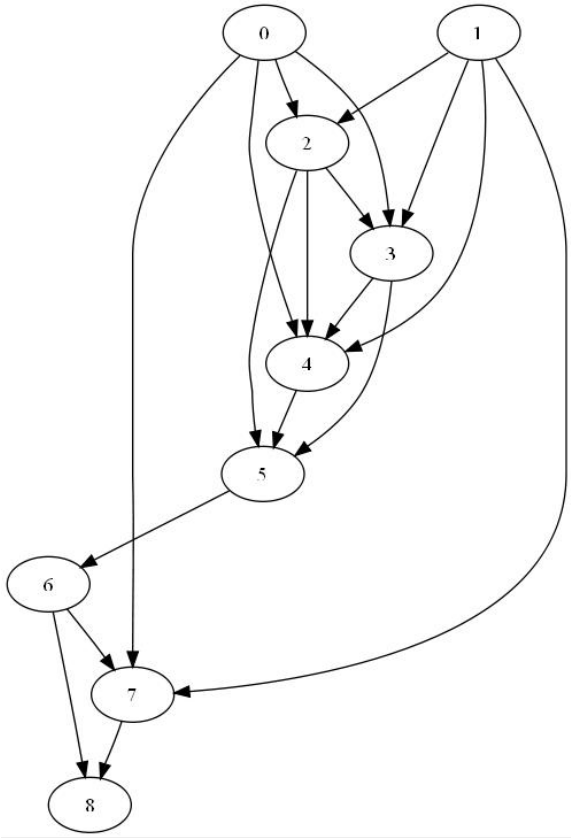

Рис. 40: еволюція 6 кроів

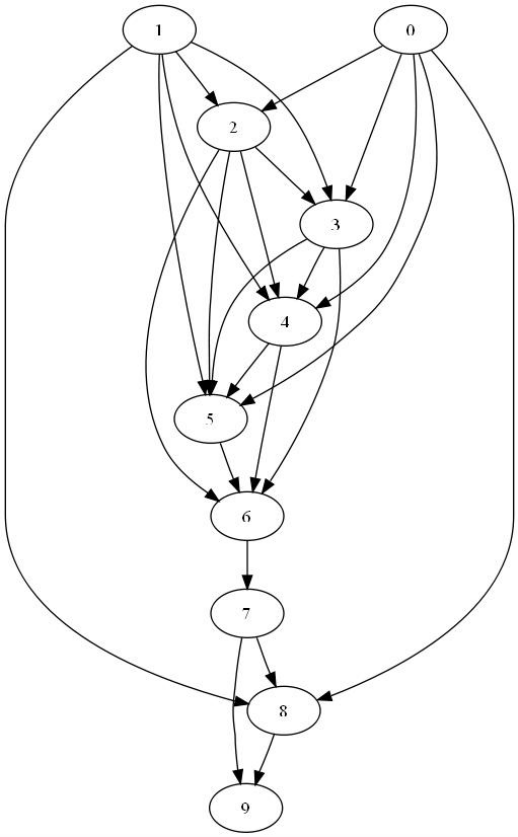

Рис. 41: еволюція 7 кроів

Порівняння об`єктивних результатів навчання наведені у таблиці

Таблиця 3

Порівняльний аналіз ефективності навчання

Мережа	Кількість нейронів	Помилка навчання (MSE)	Кількість спроб мутації
Каскадна архітектура (Рис. 34)	16	0.000001	N/A
Архітектура2-2-8-1 (Рис. 32, Рис. 33)	13	0.058941	N/A
Автоматична архітектура (Рис. 35)	20	0.103743	34

Результати, отримані в результаті навчання мережі з автоматично підібраною топологією, дещо поступаються результатами роботи запропонованих раніше мереж але є порівнюваними із ними. Згадані недоліки компенсуються тим, що навчання та проектування мережі для довільної задачі здійснюється без участі людини. Таким чином, розглянутий підхід дозволяє істотно прискорити найбільш трудомістку стадію розробки нейронних мереж - проектування їх топологій.

Задача "Detail hardness problem"

Після отримання задовільних результатів для стандартної задачі ми перейшли до експериментів на реальних даних. Для цього була взята задача із металургійної галузі.

Вона полягає у знаходженні умов обробки деталі (Рис. 42) для отримання заданих характеристик.

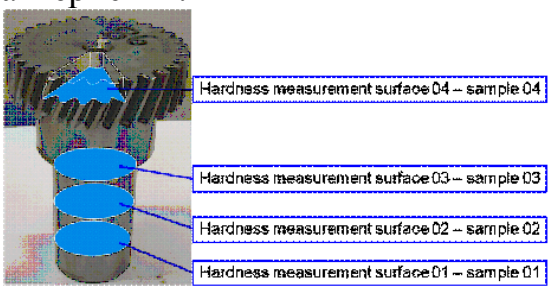

Рис. 42 : Деталь

У якості характеристик виступає твердість деталі у певних точках шестірні. Твердість осі деталі наразі не береться до уваги.

Твердість деталі вимірюється у певних точках, що мають різні полярні координати (кут та радіус). Оскільки деталь є симетричною та зваживши на умови обробки, суттєвим є лише значення радіусу точки.

Обробка деталі проходить має дві стадії:

- Нагрівання
- Охолодження

Стадія нагрівання характеризується лише температурою нагріву. Типовими значеннями є температури порядку 800..1000C.

Для охолодження деталі подається вода та повітря під певним тиском. При цьому значення тиску для подачі води та повітря можуть бути різними. Охолодження проводиться у три етапи, на кожному із яких значення тиску та температури охолоджувачів (як води, так і повітря) змінюються (Рис. 43).

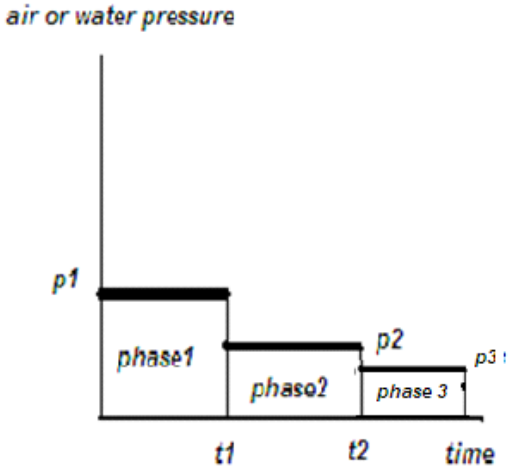

Рис. 43 : Фази охолодження

Представлення радіус-твердість.

Таким чином, шукані умови обробки складаються із десяти параметрів (Рис. 44).

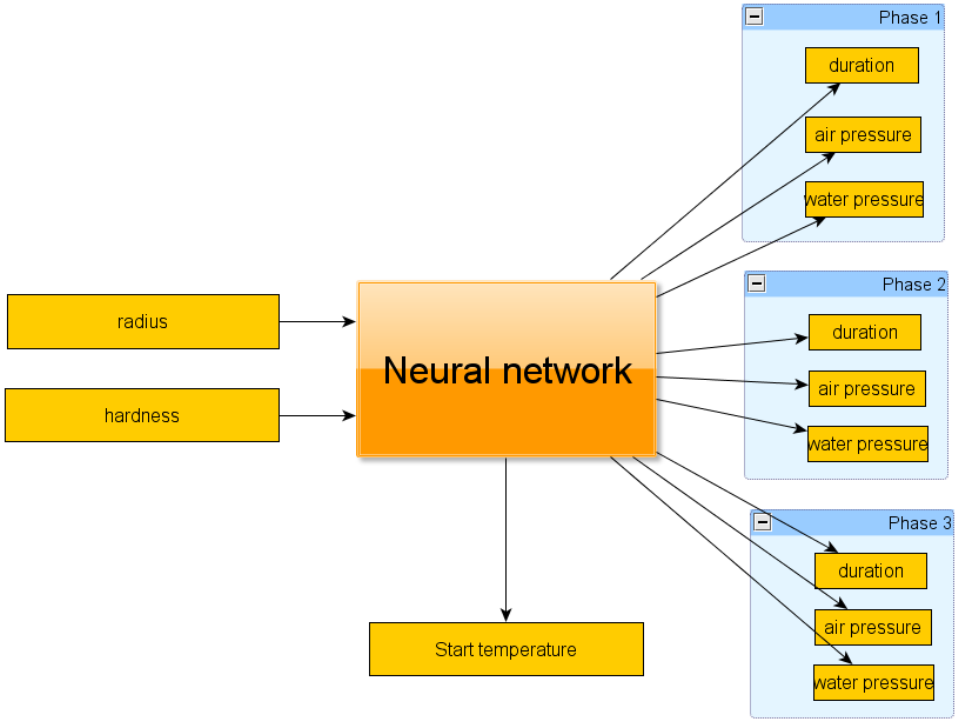

Рис. 44 : Схема входів та виходів мережі

Вхідні дані – значення твердості у точках – є результатами експериментів на металургійному заводі. Усього було проведено 11 експериментів. Кожен з них містить дані про жорсткість у 6 точках та інформацію про умови обробки (див. додаток 3).

Оскільки дані мають різну фізичну природу та діапазон значень, було проведене їхнє масштабування на діапазон [-1; +1], із яким працює нейронна мережа.

Окрім наведеної вище постановки задачі було використано попередню обробку даних (preprocessing) для покращення узагальнюючих властивостей мережі.

Поліноміальна апроксимація

Один з таких підходів – апроксимація залежності твердості від радіусу поліномом другого порядку

$h = a \cdot r^2 + b \cdot r + c$	(6)

І на вхід мережі подаються вже ці коефіцієнти (Рис. 45).

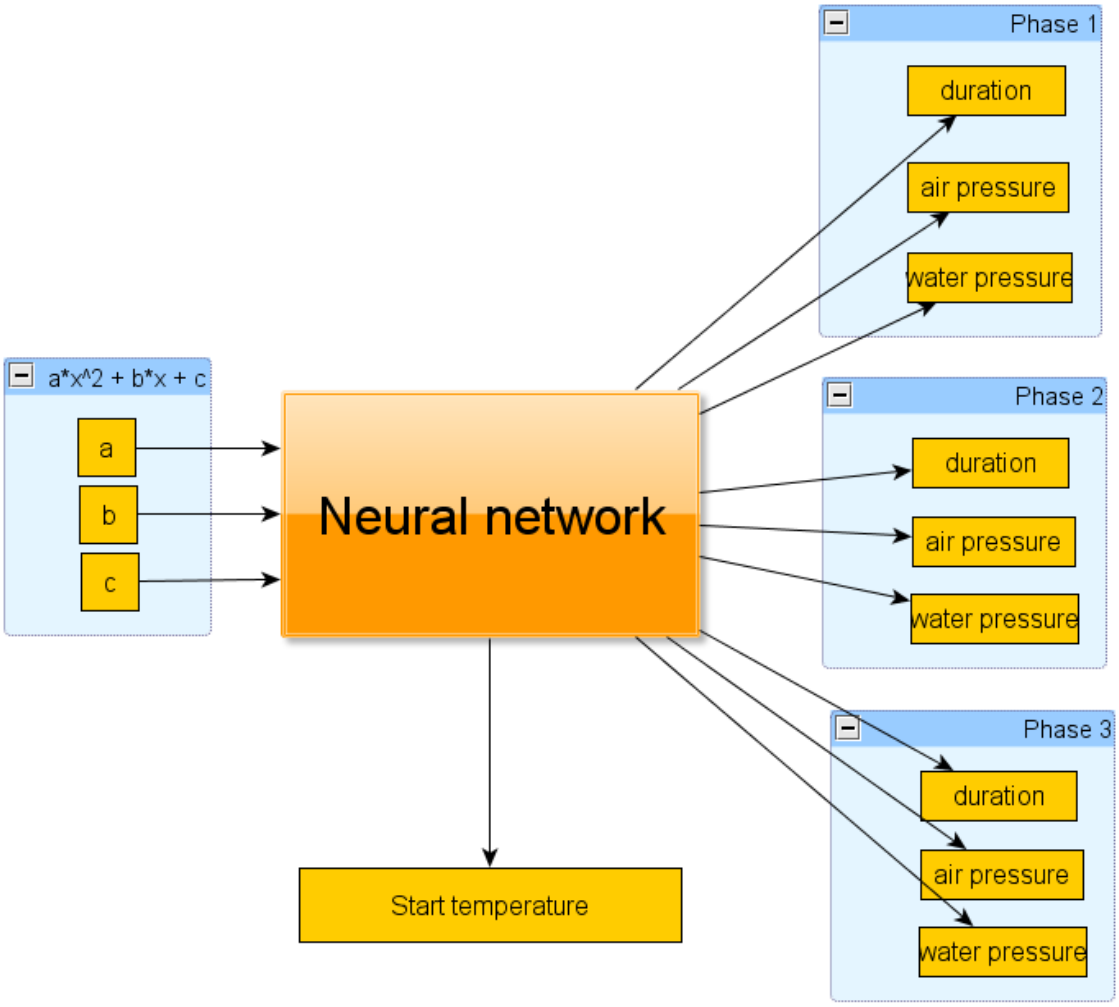

Рис. 45 : preprocessing - параболічна апроксимація

Представлення «цілий експеримент»

Оскільки один експеримент містить лише 6 точок, стає можливим розглядати його як єдине ціле та подати значення твердості в усіх точках на вхід нейронної мережі.

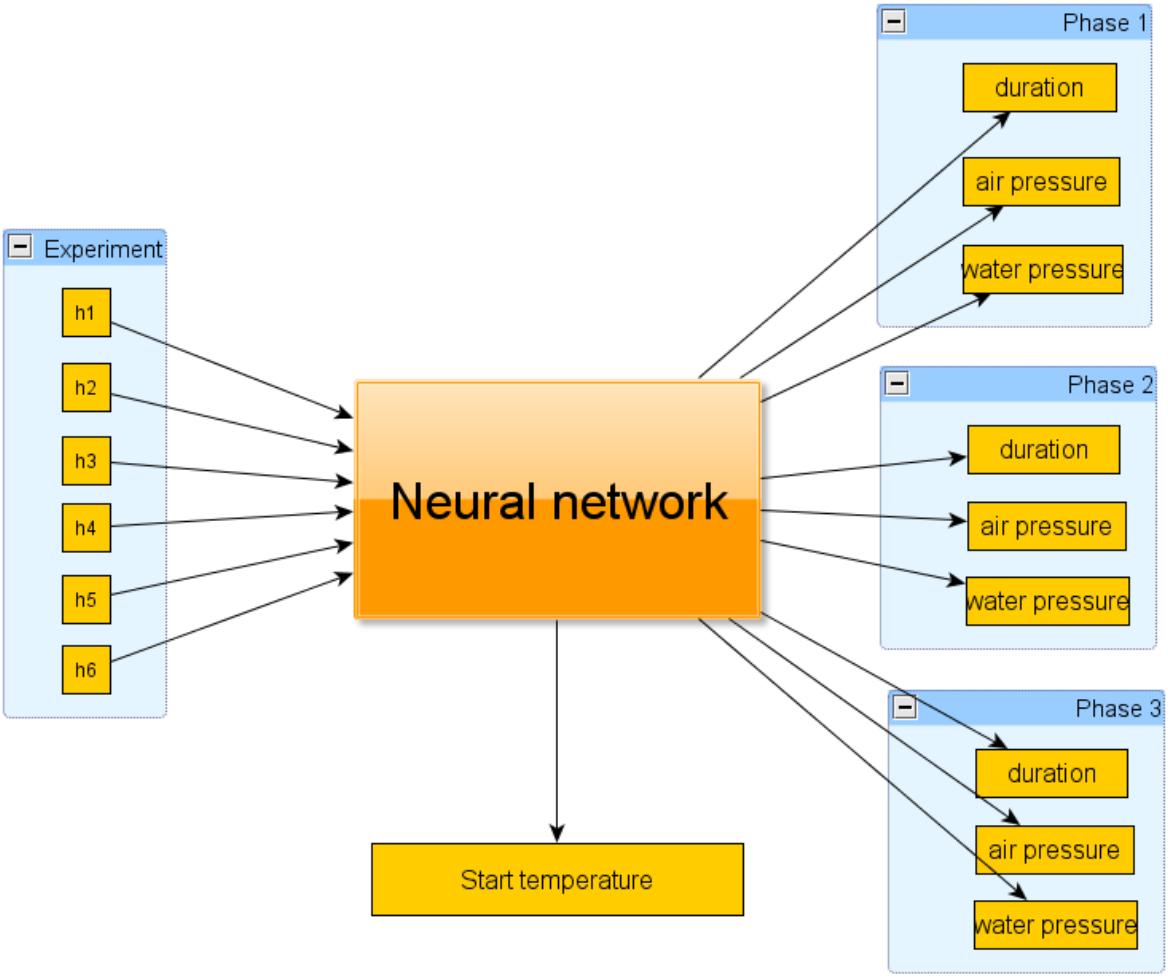

Рис. 46 : експеримент як єдине ціле

Експерименти було проведено на даних, наведених у додатку 3. Дані утворюють одинадцять експериментів. З них десять було використано для навчання нейронних мереж та один – для перевірки якості навчання.

Було проведено дві серії експериментів. В одній із них для перевірки результатів було використано нетиповий експеримент (додаток 3, експеримент №120). В іншій серії – типовий (додаток 3, експеримент №67).

У *таблиці 4* наведені умовні позначення, що надалі будуть використовуватися.

Таблиця 4 :

Умовні позначення для експериментів

Позначення	Пояснення	Архітектура мережі
1iaa	Представлення «радіус-твердість», автоматичний підбір, нетиповий експеримент.	Auto
2ian	Представлення «радіус-твердість», автоматичний підбір, типовий експеримент.	Auto
3ifa	Представлення «радіус-твердість», статична fnn мережа, нетиповий експеримент.	2-30-10
4ifn	Представлення «радіус-твердість», статична fnn мережа, типовий експеримент.	2-60-10
5paa	Представлення «коефіцієнти параболи», автоматичний підбір, нетиповий експеримент.	Auto
6pan	Представлення «коефіцієнти параболи», автоматичний підбір, типовий експеримент.	Auto
7pfa	Представлення «коефіцієнти параболи», статична fnn мережа, нетиповий експеримент.	3-50-10
8pfn	Представлення «коефіцієнти параболи», статична fnn мережа, типовий експеримент.	3-50-10
9waa	Представлення «цілий експеримент», автоматичний підбір, нетиповий експеримент.	Auto
10wan	Представлення «цілий експеримент», автоматичний підбір, типовий експеримент.	Auto
11wfa	Представлення «цілий експеримент», статична fnn мережа, нетиповий експеримент.	6-20-10
12wfn	Представлення «цілий експеримент», статична fnn мережа, типовий експеримент.	6-20-10

Нижче наведені результати цих експериментів.

У першій серії було використано атиповий експеримент. Навчена мережа повинна була відновити наступний набір характеристик металургійного процесу (*таблиця 5*) :

Таблиця 5 :

Шукані характеристики

Тиск Повітря	Тиск води	Тривалість фази	Тиск повітря	Тиск води	Тривалість фази	Тиск Повітря	Тиск Води	Тривалість фази	Початкова температура
6	0	160	3	6	20	0	0	0	850

У *таблиці 6* наведені результати навчання статичних та автоматично побудованих мереж.

Таблиця 6 :

Результати навчання – нетиповий експеримент

Id	Помилка Навчання	Помилка екзамену	Тиск повітря	Тиск води	Тривалість Фази	Тиск повітря	Тиск води	Тривалість фази	Тиск Повітря	Тиск води	Тривалість фази	Початкова температура
	N/A	N/A	6	0	160	3	6	20	0	0	0	850
1Iaa	0,72	0,89	4,16	3,05	91,09	2,31	3,17	38,14	1,16	2,34	9,96	900,7
3Ifa	0,54	2,41	6	0	160	0	0	10	3	6	20	955
5paa	0,005	1,2	3	5	59,99	3	6	54,93	0	0	0	855
7pfa	0,12	2,104	6	0	50,123	2,47	0,015	59,99	0	5,99	19,83	955
9waa	0,03	2,01	6,	0	154,83	2,99	0	28,88	3,	6,	19,99	955
11wfa	0,86	1,22	6	0	160	3	6	10,1	0	6	0	955

У другій серії було використано атиповий експеримент. Навчена мережа повинна була відновити наступний набір характеристик металургійного процесу (*таблиця 7*) :

Таблиця 7 :

Шукані характеристики

Тиск повітря	Тиск води	Тривалість фази	Тиск повітря	Тиск води	Тривалість фази	Тиск повітря	Тиск Води	Тривалість фази	Початкова температура
6	0	120	0	0	30	3	6	20	950

У *таблиці 8* наведені результати навчання статичних та автоматично побудованих мереж :

Таблиця 8 :

Результати навчання – типовий експеримент

Id	Помилка навчання	Помилка екзамену	Тиск повітря	Тиск води	Тривалість фази	Тиск повітря	Тиск води	Тривалість фази	Тиск Повітря	Тиск води	Тривалість фази	Початкова температура
	N/A	N/A	6	0	120	0	0	30	3	6	20	950
2Ian	0,64	1,092	4,56	2,39	95,29	1,26	3,02	34,35	1,51	3,02	11,33	902,4
4Ifn	0,14	0,195	6	0	140,2	0	0	34,37	3	6	16,75	955
6pan	0	0,346	4,81	1,98	80,86	0,49	0	38,38	2,39	4,79	9,32	951
8pfn	0,04	0,359	5,99	0	135,7	0,05	0	52,22	2,99	5,98	10,03	955
10wan	0,06	1,545	4,5	2,5	90	0,51	2,9	30,2	1,5	3	10,3	904,9
12wfn	0	0,242	5,99	0,005	110,5	0	0	58,76	3	5,99	17,97	954,3

Далі на Рис. 47 : 1iaaРис. 47 - Рис. 58 надано візуалізацію даних із таблиці. Враховуючи характер даних, є можливість візуалізації лише для рівнів тиску та тривалості кожної із фаз охолодження.

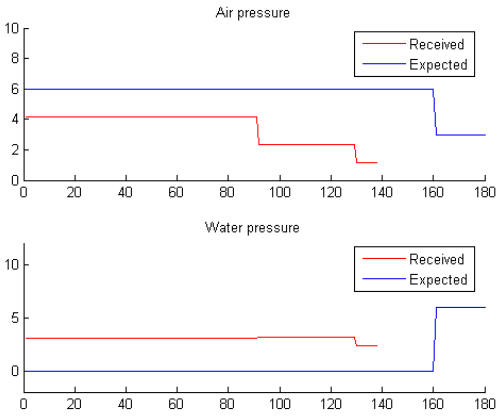

Рис. 47 : 1iaa

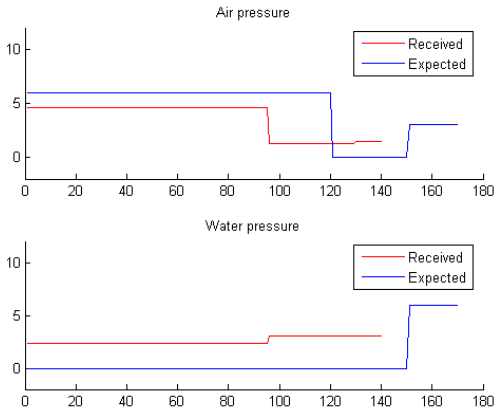

Рис. 48 : 2ian

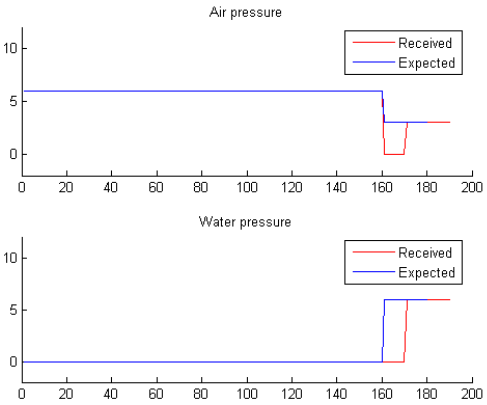

Рис. 49 : 3Ifa

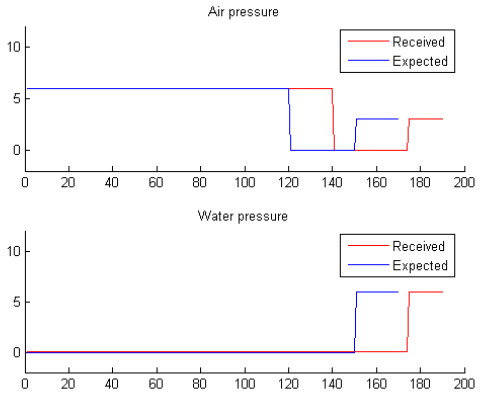

Рис. 50 : 4Ifn

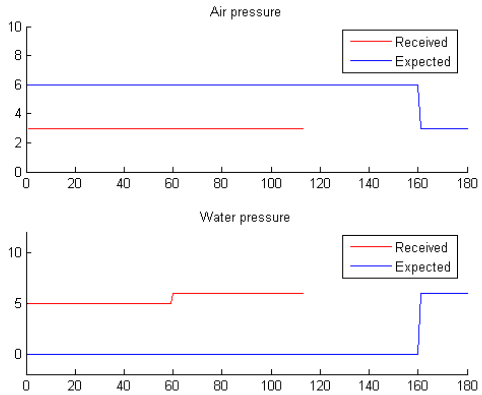

Рис. 51 : 5paa

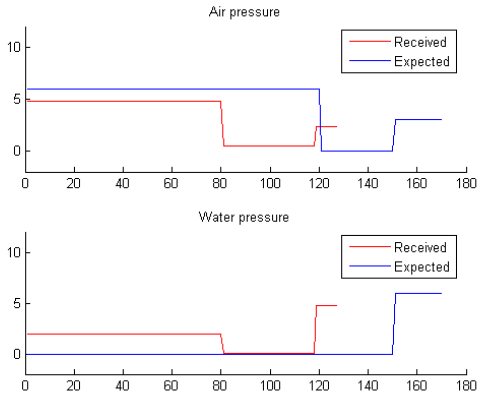

Рис. 52 : 6pan

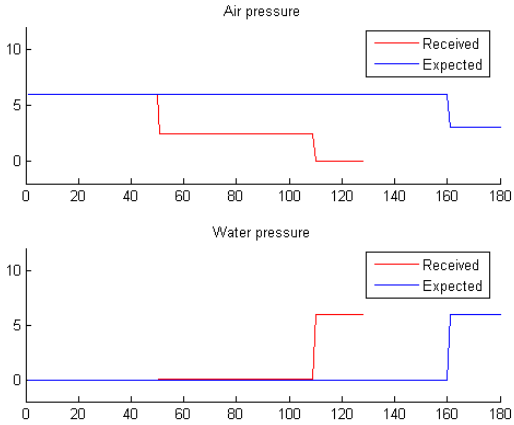

Рис. 53 : 7pfa

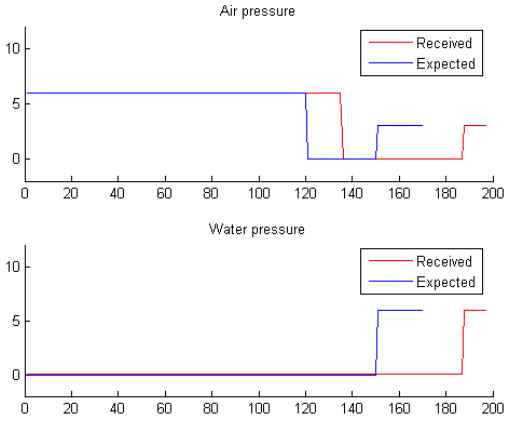

Рис. 54 : 8pfn

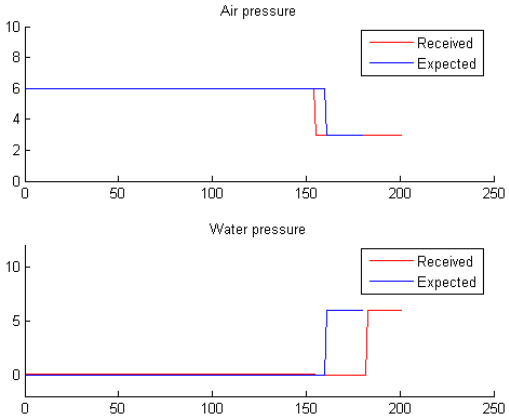

Рис. 55 : 9waa

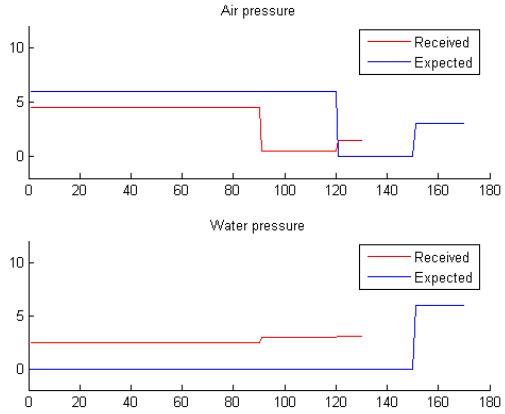

Рис. 56 : 10wan

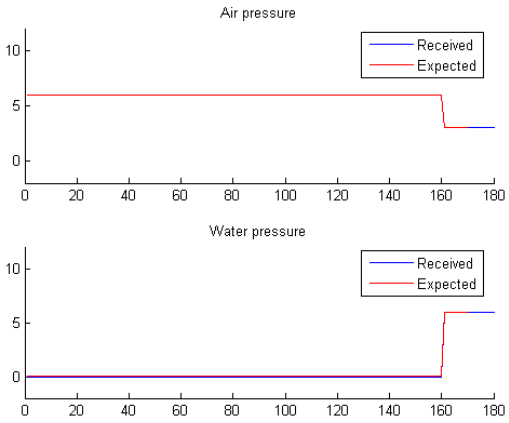

Рис. 57 : 11wfa

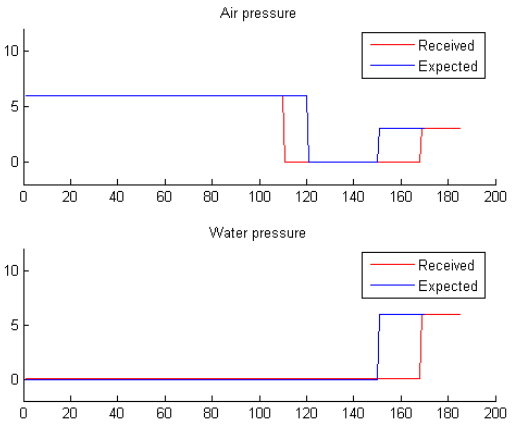

Рис. 58 : 12wfn

Аналізуючи наведені вище результати експериментів, можна дійти висновку, що для розв’язання розглянутої задачі мережі прямого розповсюдження є більш пристосованими, ніж мережі із каскадною архітектурою.

Тим-не-менше, на представленні «**цілий експеримент**» було отримано порівнювані результати за допомогою автоматичного підбору.

Таким чином, запропонована у даній роботі методика не є достатньо ефективною для даної задачі. Вона потребує більш тонкого налаштування та розширення множини операторів мутації.

Однак, порівнювана із **fnn** якість навчання в одному з експериментів свідчить про її перспективність.

Висновки

Таким чином, в результаті даної роботи був отриманий інструмент для побудови та подальшого навчання нейронних мереж прямого розповсюдження з перехресними зв'язками. Для представлення таких мереж була розроблена компактна схема кодування, заснована на ***L-системах***, придатна для подальшого застосування в еволюційних алгоритмах. Також була створена бібліотека для перетворення нейромереж, представлених у вигляді G0L граматики або матриці суміжності графа топології мережі у формат ***Matlab neural network toolbox***.

Результати, отримані в результаті навчання мережі з автоматично підібраною топологією, дещо поступаються результатами роботи запропонованих раніше мереж для задачі «***Two spiral problem***». Ми вважаємо, що це обумовлено наявністю обмежень максимальної кількості нейронів у мережі і кількістю переписувань.

На реальній задачі «***Detail hardness problem***» з області металургії наша методика показала порівнювані із **fnn** результати. Це зумовлено тим, що каскадні мережі менш придатні для розв’язання цієї задачі, ніж **fnn** мережі. Тим-не-менше, на одному із запропонованих представлень було отримано порівнювані із **fnn** результати, що свідчить про перспективність даної методики.

Результати автоматичного підбору також можуть бути покращені за рахунок більш детального вивчення архітектури, запропонованої нашою системою, більш вдалого вибору початкової архітектури мережі в залежності від розв'язуваної задачі або використанням більш складних і ефективних еволюційних алгоритмів.

Згадані недоліки компенсуються тим, що навчання та проектування мережі для довільної задачі здійснюється без участі людини. Розглянутий в даній роботі підхід дозволяє істотно прискорити найбільш трудомістку стадію розробки нейронних мереж – проектування їхніх топологій.

ЛІТЕРАТУРА

1. **Boers E. J. W., Sprinkhuizen-Kuyper I.** Combined Biological Metaphors // Advances in the Neural Information Processing Systems. MIT Press. – 2001. – P. 53-183.
2. **Fahlman S. E. Lebiere C**. The Cascade-Correlation Learning Architecture. // CMU-CS. – 1991. – P. 90-100.
3. Freeware academic implementation of the Cascade Correlation Algorithm for Matlab and Octave, Lab. Associado de Computacao e Matematica Aplicada - LAC/INPE [електронний ресурс] / **Silva, J.D.S**. – 2006. – Режим доступу: http://www.lac.inpe.br/~demisio/downloads.html
4. **Springer P.** A Concurrent Implementation of the Cascade-Correlation Algorithm, Using the Time Warp Operating System // Jet Propulsion Laboratory California Institute of Technology
5. **Yuret D**. From Genetic Algorithms To Efficient Optimization. // Technical Report No. 1569. / Massachusetts institute of technology. Artificial intelligence laboratory , 1994
6. Бібліотека boost [електронний ресурс] / Режим доступу : http://www.boost.org/
7. Лексичний аналізатор flex [електронний ресурс] / Режим доступу : *http://flex.sourceforge.net/*
8. Бібліотека Matlab для роботи із форматом XML [електронний ресурс] / Режим доступу : http://www.geodise.org/toolboxes/generic/xml_toolbox.htm
9. Синтаксичний аналізатор GNU Bison [електронний ресурс] / Режим доступу : *http://www.gnu.org/software/bison/*
10. Проект із відкритим кодом Python robotic [електронний ресурс] / Режим доступу : http://pyrorobotics.org/?page=Pyro
11. Проект із відкритим кодом Python robotic – опис задачі «Two spirals problem» [електронний ресурс] / Режим доступу : http://emergent.brynmawr.edu/emergent/TwoSpiralsProblem
12. Відкрита енциклопедія «Вікіпедія» [електронний ресурс] / Режим доступу : http://en.wikipedia.org/wiki/Main_Page
13. Електронна довідка функцій Matlab [електронний ресурс] / Режим доступу : http://www.mathworks.com/access/helpdesk/help/toolbox/nnet/
14. **Wu Youshou**, **Zhao Mingsheng**, A neuron model with trainable activation function and its MFNN supervised learning. – 2001
15. **Кузнецов К.А**., **Додатко А.В**. Методы адаптивного проектирования нейронных сетей прямого распространения с каскадной архитектурой. // XVI конференция по автоматическому управлению «Автоматика-2009». – Черновцы: Книги XXI, 2009. – с332-333

16. **Кузнецов К.А.**, **Додатко А.В.** Методы адаптивного проектирования нейронных сетей прямого распространения с каскадной архитектурой. // Системные технологии. Региональный межвузовский сборник научных работ. – Выпуск 2(67). – Днепропетровск, 2010. – С.180 – 190.
17. **Комашинский Д.И, Смирнов Д.А.** Нейронные сети и их применение в системах управления и связи. // М. : Горячая линия – телеком, 2003. – 94с.
18. **Бодянский Е.В., Руденко О.Г.** Нейронные сети : архитектуры, обучение, применения. // Харьков : ТЕЛЕТЕХ, 2004. – 369с. : ил.
19. **Осовский С.** Нейронные сети для обработки информации. / пер. с английского И.Д.Рудинского. – М. Финансы и статистика, 2002. – 344с : ил.
20. **Рутковская Д., Пилиньский М., Рутковский Л.** Нейронные сети, генетические алгоритмы и нечёткие системы / пер. с польск. И.Д.Рудинского. – М. : Горячая линия – Телеком, 2006. – 456с.: ил.
21. **Вороновский Г.К., и др.** Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / Г.К. Вороновский, К.В. Махотило, С.Н. Петрашев, С. А. Сергеев. – Х. : ОСНОВА, 1997. – 112с.
22. **Хайкин С.** Нейронные сети : полный курс, 2-е издание. / Пер. с англ. – М. : Издательский дом «Вильямс», 2008. – 1104с. : ил. – Парал. тит. англ.

ДОДАТКИ

Додаток 1 : Код для генерації даних для задачі „Two Spiral Problem”

Код на мові „C”

```
void printSet  ( int density, double maxRadius )
{
   int points,      /* Number of interior data points to generate */
   i;          /* Indexing variable */
   double x,        /* x coordinate */
   y,       /* y coordinate */
   angle,    /* Angle to calculate */
   radius;   /* Radius at current iteration */

   printf  ("$TRAIN\n\n");
   points = ( 96 * density );
   for  ( i = 0 ; i <= points ; i++ )
   {

      /* Angle is based on the iteration * PI/16, divided by point density */
      angle = ( i * PI ) / ( 16.0 * density );

      /* Radius is the maximum radius * the fraction of iterations left */
      radius = maxRadius * ( ( 104 * density ) - i ) / ( 104 * density );

      /* x and y are based upon cos and sin of the current radius */
      x = radius * cos( angle );
      y = radius * sin( angle );

      printf ("%8.5f, %8.5f  =>  +;\n", x, y );
      printf ("%8.5f, %8.5f  =>  -;\n", -x, -y );
   }
}
```

Код на мові „Matlab”

```
function [x1, y1] = CreateSpiral(maxRadius, density, count, noiseMode)
   x1 = []; y1 = [];

   angle = 0;
   radius = 0;
   for index = 1 : count
      angle = (index * pi) / (16 * density);

      tmp = 104 * density;
      radius = maxRadius * (tmp - index) / tmp;

      nX = radius * cos(angle);
      nY = radius * sin(angle);

      [nX, nY] = AddNoise(nX, nY, noiseMode);

      x1 = [x1 nX];
      y1 = [y1 nY];
   end

return
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function [nX, nY] = AddNoise(x, y, noiseMode)

   NOISE = 0.12;
   xFlag = 'x';
   yFlah = 'y';

   nX = x;
   nY = y;

   noiseX = (2 * rand() - 1) * NOISE;
   noiseY = (2 * rand() - 1) * NOISE;

   b_xNoise = ~isempty(strfind(noiseMode, xFlag));
   b_yNoise = ~isempty(strfind(noiseMode, yFlah));


   if (b_xNoise)
      nX = nX + noiseX;
   end

   if (b_yNoise)
      nY = nY + noiseY;
   end

return
end
```

Додаток 2 : Дані для задачі „Two Spiral Problem”

1.00000	0.50000	1.0
0.00000	0.50000	0.0
0.98568	0.60466	1.0
0.01432	0.39534	0.0
0.95306	0.70330	1.0
0.04694	0.29670	0.0
0.90374	0.79225	1.0
0.09626	0.20775	0.0
0.83995	0.86829	1.0
0.16005	0.13171	0.0
0.76443	0.92873	1.0
0.23557	0.07127	0.0
0.68030	0.97156	1.0
0.31970	0.02844	0.0
0.59098	0.99550	1.0
0.40902	0.00450	0.0
0.50000	1.00000	1.0
0.50000	0.00000	0.0
0.41089	0.98528	1.0
0.58911	0.01472	0.0
0.32705	0.95232	1.0
0.67295	0.04768	0.0
0.25159	0.90274	1.0
0.74841	0.09726	0.0
0.18724	0.83882	1.0
0.81276	0.16118	0.0
0.13623	0.76331	1.0
0.86377	0.23669	0.0
0.10024	0.67938	1.0
0.89976	0.32062	0.0
0.08034	0.59043	1.0
0.91966	0.40957	0.0
0.07692	0.50000	1.0
0.92308	0.50000	0.0
0.08977	0.41160	1.0
0.91023	0.58840	0.0
0.11801	0.32859	1.0
0.88199	0.67141	0.0
0.16022	0.25404	1.0
0.83978	0.74596	0.0
0.21444	0.19064	1.0
0.78556	0.80936	0.0
0.27831	0.14056	1.0
0.72169	0.85944	0.0
0.34914	0.10542	1.0
0.65086	0.89458	0.0
0.42403	0.08623	1.0
0.57597	0.91377	0.0
0.50000	0.08333	1.0
0.50000	0.91667	0.0
0.57410	0.09645	1.0
0.42590	0.90355	0.0
0.64351	0.12468	1.0

0.35649	0.87532	0.0
0.70567	0.16655	1.0
0.29433	0.83345	0.0
0.75837	0.22011	1.0
0.24163	0.77989	0.0
0.79981	0.28298	1.0
0.20019	0.71702	0.0
0.82869	0.35251	1.0
0.17131	0.64749	0.0
0.84422	0.42583	1.0
0.15578	0.57417	0.0
0.84615	0.50001	1.0
0.15385	0.49999	0.0
0.83479	0.57215	1.0
0.16521	0.42785	0.0
0.81092	0.63953	1.0
0.18908	0.36047	0.0
0.77582	0.69966	1.0
0.22418	0.30034	0.0
0.73117	0.75044	1.0
0.26883	0.24956	0.0
0.67895	0.79015	1.0
0.32105	0.20985	0.0
0.62142	0.81759	1.0
0.37858	0.18241	0.0
0.56096	0.83204	1.0
0.43904	0.16796	0.0
0.49999	0.83333	1.0
0.50001	0.16667	0.0
0.44090	0.82182	1.0
0.55910	0.17818	0.0
0.38593	0.79833	1.0
0.61407	0.20167	0.0
0.33706	0.76416	1.0
0.66294	0.23584	0.0
0.29602	0.72097	1.0
0.70398	0.27903	0.0
0.26415	0.67072	1.0
0.73585	0.32928	0.0
0.24238	0.61560	1.0
0.75762	0.38440	0.0
0.23123	0.55791	1.0
0.76877	0.44209	0.0
0.23077	0.49999	1.0
0.76923	0.50001	0.0
0.24066	0.44411	1.0
0.75934	0.55589	0.0
0.26015	0.39236	1.0
0.73985	0.60764	0.0
0.28814	0.34663	1.0
0.71186	0.65337	0.0
0.32323	0.30849	1.0
0.67677	0.69151	0.0

0.36378	0.27914	1.0
0.63622	0.72086	0.0
0.40801	0.25940	1.0
0.59199	0.74060	0.0
0.45405	0.24969	1.0
0.54595	0.75031	0.0
0.50001	0.25000	1.0
0.49999	0.75000	0.0
0.54409	0.25991	1.0
0.45591	0.74009	0.0
0.58464	0.27866	1.0
0.41536	0.72134	0.0
0.62020	0.30513	1.0
0.37980	0.69487	0.0
0.64958	0.33796	1.0
0.35042	0.66204	0.0
0.67189	0.37558	1.0
0.32811	0.62442	0.0
0.68655	0.41629	1.0
0.31345	0.58371	0.0
0.69333	0.45835	1.0
0.30667	0.54165	0.0
0.69231	0.50001	1.0
0.30769	0.49999	0.0
0.68390	0.53963	1.0
0.31610	0.46037	0.0
0.66878	0.57574	1.0
0.33122	0.42426	0.0
0.64790	0.60707	1.0
0.35210	0.39293	0.0
0.62238	0.63259	1.0
0.37762	0.36741	0.0
0.59348	0.65157	1.0
0.40652	0.34843	0.0
0.56255	0.66361	1.0
0.43745	0.33639	0.0
0.53095	0.66857	1.0
0.46905	0.33143	0.0
0.49999	0.66667	1.0
0.50001	0.33333	0.0
0.47092	0.65835	1.0
0.52908	0.34165	0.0
0.44480	0.64435	1.0
0.55520	0.35565	0.0
0.42254	0.62558	1.0

0.57746	0.37442	0.0
0.40481	0.60312	1.0
0.59519	0.39688	0.0
0.39207	0.57812	1.0
0.60793	0.42188	0.0
0.38451	0.55182	1.0
0.61549	0.44818	0.0
0.38212	0.52540	1.0
0.61788	0.47460	0.0
0.38462	0.50000	1.0
0.61538	0.50000	0.0
0.39155	0.47663	1.0
0.60845	0.52337	0.0
0.40228	0.45615	1.0
0.59772	0.54385	0.0
0.41606	0.43923	1.0
0.58394	0.56077	0.0
0.43201	0.42634	1.0
0.56799	0.57366	0.0
0.44925	0.41772	1.0
0.55075	0.58228	0.0
0.46689	0.41338	1.0
0.53311	0.58662	0.0
0.48406	0.41316	1.0
0.51594	0.58684	0.0
0.50000	0.41667	1.0
0.50000	0.58333	0.0
0.51407	0.42338	1.0
0.48593	0.57662	0.0
0.52576	0.43263	1.0
0.47424	0.56737	0.0
0.53473	0.44370	1.0
0.46527	0.55630	0.0
0.54080	0.45581	1.0
0.45920	0.54419	0.0
0.54397	0.46817	1.0
0.45603	0.53183	0.0
0.54442	0.48007	1.0
0.45558	0.51993	0.0
0.54244	0.49086	1.0
0.45756	0.50914	0.0
0.53846	0.50000	1.0
0.46154	0.50000	0.0

Додаток 3 : Дані для задачі «Detail hardness problem»

Додаток 3.1 Представлення «радіус-твердість»

Pa1	Pw1	Dt1	Pa2	Pw2	Dt2	Pa3	Pw3	Dt3	T		r	h
3	5	20	3	0	50	0	0	0	950		0	629
3	5	20	3	0	50	0	0	0	950		0,2	630
3	5	20	3	0	50	0	0	0	950		0,4	626
3	5	20	3	0	50	0	0	0	950		0,6	617
3	5	20	3	0	50	0	0	0	950		0,8	611
3	5	20	3	0	50	0	0	0	950		1	603
6	0	120	0	0	30	3	6	20	950		0	578
6	0	120	0	0	30	3	6	20	950		0,2	581
6	0	120	0	0	30	3	6	20	950		0,4	580
6	0	120	0	0	30	3	6	20	950		0,6	578
6	0	120	0	0	30	3	6	20	950		0,8	576
6	0	120	0	0	30	3	6	20	950		1	567
6	0	120	0	0	20	3	6	20	950		0	489
6	0	120	0	0	20	3	6	20	950		0,2	500
6	0	120	0	0	20	3	6	20	950		0,4	505
6	0	120	0	0	20	3	6	20	950		0,6	497
6	0	120	0	0	20	3	6	20	950		0,8	484
6	0	120	0	0	20	3	6	20	950		1	474
6	0	120	0	0	60	3	6	3	950		0	539
6	0	120	0	0	60	3	6	3	950		0,2	539
6	0	120	0	0	60	3	6	3	950		0,4	534
6	0	120	0	0	60	3	6	3	950		0,6	517
6	0	120	0	0	60	3	6	3	950		0,8	523
6	0	120	0	0	60	3	6	3	950		1	510
6	0	160	2	0	30	3	6	20	950		0	572
6	0	160	2	0	30	3	6	20	950		0,2	603
6	0	160	2	0	30	3	6	20	950		0,4	611
6	0	160	2	0	30	3	6	20	950		0,6	613
6	0	160	2	0	30	3	6	20	950		0,8	598
6	0	160	2	0	30	3	6	20	950		1	590
6	0	120	0	0	10	3	6	20	950		0	566
6	0	120	0	0	10	3	6	20	950		0,2	562
6	0	120	0	0	10	3	6	20	950		0,4	544
6	0	120	0	0	10	3	6	20	950		0,6	538
6	0	120	0	0	10	3	6	20	950		0,8	522
6	0	120	0	0	10	3	6	20	950		1	500
6	0	120	0	0	30	3	6	3	950		0	579
6	0	120	0	0	30	3	6	3	950		0,2	589

6	0	120	0	0	30	3	6	3	950		0,4	573
6	0	120	0	0	30	3	6	3	950		0,6	563
6	0	120	0	0	30	3	6	3	950		0,8	555
6	0	120	0	0	30	3	6	3	950		1	545
6	0	160	0	0	30	3	6	20	955		0	577
6	0	160	0	0	30	3	6	20	955		0,2	601
6	0	160	0	0	30	3	6	20	955		0,4	600
6	0	160	0	0	30	3	6	20	955		0,6	592
6	0	160	0	0	30	3	6	20	955		0,8	572
6	0	160	0	0	30	3	6	20	955		1	553
6	0	160	2	0	30	3	6	20	955		0	659
6	0	160	2	0	30	3	6	20	955		0,2	680
6	0	160	2	0	30	3	6	20	955		0,4	676
6	0	160	2	0	30	3	6	20	955		0,6	664
6	0	160	2	0	30	3	6	20	955		0,8	645
6	0	160	2	0	30	3	6	20	955		1	621
6	0	160	2	0	30	3	6	20	955		0	596
6	0	160	2	0	30	3	6	20	955		0,2	637
6	0	160	2	0	30	3	6	20	955		0,4	653
6	0	160	2	0	30	3	6	20	955		0,6	664
6	0	160	2	0	30	3	6	20	955		0,8	656
6	0	160	2	0	30	3	6	20	955		1	648
6	0	160	3	6	20	0	0	0	850		0	606
6	0	160	3	6	20	0	0	0	850		0,2	648
6	0	160	3	6	20	0	0	0	850		0,4	638
6	0	160	3	6	20	0	0	0	850		0,6	621
6	0	160	3	6	20	0	0	0	850		0,8	612
6	0	160	3	6	20	0	0	0	850		1	597

Додаток 3.2 : Представлення «Коефіцієнти параболи»

Шукані дані :

#	Pa	Pw	Dt1	Pa	Pw	Dt2	Pa	Pw	Dt3	T
1	3	5	20	3	0	50	0	0	0	950
2	6	0	120	0	0	30	3	6	20	950
4	6	0	120	0	0	20	3	6	20	950
4	6	0	120	0	0	60	3	6	3	950
5	6	0	160	2	0	30	3	6	20	950
6	6	0	120	0	0	10	3	6	20	950
7	6	0	120	0	0	30	3	6	3	950
8	6	0	160	0	0	30	3	6	20	955
9	6	0	160	2	0	30	3	6	20	955
10	6	0	160	2	0	30	3	6	20	955
11	6	0	160	3	6	20	0	0	0	850

Вихідні дані : ($y = ax^2 + bx + c$)

	1	2	3	4	5	6	7	8	9	10	11
A	-23,66	-28,57	-79,01	-9 375	-128,1	-36,6	-30,35	-129,9	-127,2	-152,2	125,4
B	-4 339	18,28	60,3	-20,62	139,1	-28,53	-9 928	99,19	83,37	199	101,1
C	630,1	578	490,3	540,7	575,2	566,3	583,4	580,5	662,4	598,6	615,7

Додаток 3.3 : Представлення «цілий експеримент»

Шукані дані :

#	Pa	Pw	Dt1	Pa	Pw	Dt2	Pa	Pw	Dt3	T
1	3	5	20	3	0	50	0	0	0	950
2	6	0	120	0	0	30	3	6	20	950
4	6	0	120	0	0	20	3	6	20	950
4	6	0	120	0	0	60	3	6	3	950
5	6	0	160	2	0	30	3	6	20	950
6	6	0	120	0	0	10	3	6	20	950
7	6	0	120	0	0	30	3	6	3	950
8	6	0	160	0	0	30	3	6	20	955
9	6	0	160	2	0	30	3	6	20	955
10	6	0	160	2	0	30	3	6	20	955
11	6	0	160	3	6	20	0	0	0	850

Вихідні дані

	1	2	3	4	5	6	7	8	9	10	11
H1	629	578	489	539	572	566	579	577	659	596	606
H2	630	581	500	539	603	562	589	601	680	637	648
H3	626	580	505	534	611	544	573	600	676	653	638
H4	617	578	497	517	613	538	563	592	664	664	621
H5	611	576	484	523	598	522	555	572	645	656	612
H6	603	567	474	510	590	500	545	553	621	648	597