МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет інформаційних технологій
Кафедра інтелектуальних технологій

ВИПУСКНА КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

НА ТЕМУ

Розробка програмного модулю для діагностики несправностей електричних двигунів на основі нейронних мереж

Галузь знань **12 «Інформаційні технології»**
Спеціальність **122 «Комп'ютерні науки»**
Освітня програма **«Комп'ютерні науки»**
Освітній рівень: бакалавр

Виконав: студент 4 курсу, групи КН- 41
___Мартус А. А._____
(прізвище та ініціали)

Керівник _Андрійчук О. В._____
(прізвище та ініціали)
кандидат технічних наук, асистент
(науковий ступінь, звання)

Випускна кваліфікаційна робота бакалавра допущена до захисту
рішенням кафедри *інтелектуальних технологій*
Протокол № від р.
зав. кафедри_____ доц. Іларіонов О.Є.

Київ – 2022

Анотація

В даній роботі вивчаються можливості використання нейронних мереж для класифікації сигналів тахометра, що надходять від обертового двигуна. Згідно із фізичною сутністю, сигнали моделюються як зашумлені гармонічні сигнали за наявності різних за своєю природою перешкод: вібрацій, попадання чужорідних тіл, проблем із подачею палива тощо. На основі проведеного аналізу обґрунтовується згортка багатокомпонентних векторів сигналів у характеристичні 16-компонентні вектори. В роботі запропоновано та реалізовано послідовність етапів навчання та верифікації нейронних мереж, а також принципи створення та навчання мета структури – каскадного класифікатора побудованого на наборі нейромереж, здатного розрізняти позитивний сигнал та декілька неполадок. Побудований із 15 нейромереж каскад, що здатний розпізнавати штатну роботу та 5 неполадок, показав наступні результати: чутливість 1,000; селективність 0,988; точність 0,990; коефіцієнт Метьюза 0,965. Таким чином, отримана система демонструє дуже високу якість класифікації, що вказує на перспективність запропонованого підходу.

Ключові слова: багатошарова нейронна мережа, згортка сигналів, навчання нейронної мережі, верифікація нейронної мережі, метаструктура із набору нейромереж

Abstract

This paper examines the application of ANN techniques to classify tachometer signals from a rotating engine. According to the physical essence, the input signals are modeled as harmonic signals with noise caused by various distortions, such as: vibration, foreign bodies, fuel problems, and so on. The convolution of multicomponent signal vectors into the characteristic 16-component vectors is substantiated. The sequence of stages of ANN training and verification, as well as the principles of creation and training the metastructure - a cascade classifier built on a set of ANNs, able to distinguish between a positive signal and several distortions - are proposed and implemented. The cascade built over 15 neural networks, able to recognize normal work and 5 distortions, showed the following results: sensitivity is 1,000; selectivity - 0.988; accuracy - 0.990; Matthews coefficient - 0.965. Thus, the developed system demonstrates a very high quality of classification and indicates the feasibility of the proposed approach

Keywords: multilayer neural network, signal convolution, ANN training, ANN verification, metastructure over a set of ANNs

Зміст

ВСТУП

Актуальність теми дослідження. Перехід до нового технологічного укладу, що отримав назву "промисловість 4.0" супроводжується значним ускладненням виробництва із впровадженням інтелектуальних систем управління процесами, контролю якості та безпеки. Основними інструментами, що забезпечують розвиток нових технологій, стають нейронні мережі, котрі здатні працювати у реальному часі з великою точністю і донавчатися у процесі використання. Переваги нейромереж роблять їх привабливими для вирішення таких завдань, як:

- планування і прогнозування;
- проектування АСУ;
- управління якістю;
- управління маніпуляторами та робототехнікою;
- забезпечення безпеки виробництва: виявлення несправностей та попередження аварійних ситуацій;
- управління процесами: оптимізація режимів виробничих процесів;
- моніторинг та візуалізація диспетчерської інформації.

Наприклад, нейронна мережа на підприємствах Intel здатна ідентифікувати брак при виробництві мікросхем та здатна забракувати несправний чіп із точністю 99,5%[1]. Інший приклад – перевірка якості бетону спеціалістами із National Institute of Standards and Technology (NIST), які створили систему, що подає звукові хвилі та приймає відбитий сигнал на глибину до півметра, а дані аналізує нейромережа[2].

В даній роботі вивчаються можливості використання нейронних мереж для класифікації сигналів тахометра, що надходять від обертового двигуна.

Предметом дослідження є багатошарові нейронні мережі.

[1] Ghahramani, M.; Qiao, Y.; Zhou, M.; Hagan, A.O.; Sweeney, J. AI-based modeling and data-driven evaluation for smart manufacturing processes. IEEE/CAA J. Autom. Sin. 2020, 7, 1026–1037.

[2] Patan K., Korbicz J. Application of Dynamic Neural Networks in an Industrial Plant // 4th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes 2000 (SAFEPROCESS 2000), Budapest, Hungary, 14-16 June 2000 // https://doi.org/10.1016/S1474-6670(17)37358-5

Об'єктом дослідження виступають особливості застосування нейронних мереж для класифікації сигналів від працюючого двигуна.

Метою роботи є розробка архітектури та програмна реалізація системи штучного інтелекту на основі багатошарових нейронних мереж із застосуванням згортки вхідних сигналів та метаструктури (каскаду) із навчених нейромереж, що здатна розрізняти штатну роботу та декілька передаварійних ситуацій із високою точністю.

Згідно із визначеними предметом, об'єктом та метою, перед дослідженням поставлені наступні *завдання*:

- провести аналіз принципів побудови багатошарових нейронних мереж, надати огляд застосування систем штучного інтелекту на сучасних виробництвах;
- обґрунтувати структуру згортки сигналу від детектору обертів двигуна у характерний вектор для навчання нейронних мереж;
- визначити послідовність етапів навчання та верифікації нейронних мереж, схему побудови каскаду із декількох нейромереж для сумісної роботи;
- реалізувати запропоновані архітектурні рішення у програмному комплексі;
- провести тестування програмного комплексу, проаналізувати отримані результати;
- зробити висновки на основі проведеної роботи, визначити напрямки подальших досліджень.

Теоретико-методологічною основою розробки є загальнонаукові та спеціальні методи розрахунків та проектування, розробки методик дослідження та впровадження програмних комплексів.

Інформаційну базу дослідження становлять нормативні документи, стандарти, статті, монографії та інші ресурси англійською, українською та іншими мовами.

Наукова новизна даної роботи полягає в обґрунтуванні згортки сигналів від детектору у характеристичний вектор, виходячи із фізичної природи сигналу, запропонованій та реалізованій послідовності навчання та верифікації нейронних мереж, а також у розробці та реалізації каскаду із декількох навчених нейронних мереж для класифікації сигналів із високою точністю.

Науково-практичне значення. Дане дослідження буде корисним для фахівців у галузі прикладної математики, розробки інформаційних систем, а також менеджерів різних галузей виробництва, що планують впровадження систем штучного інтелекту.

Структура роботи відповідає меті та завданням дослідження. Робота складається із вступу, трьох розділів, висновків, списку використаних джерел та додатків.

РОЗДІЛ 1. ОГЛЯД ПРОБЛЕМНОЇ ОБЛАСТІ, ОСНОВНІ ПРОПОЗИЦІЇ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Аналіз принципів побудови та практики застосування багатошарових нейронних мереж на сучасному виробництві

Базовими елементами кожної нейронної системи є відносно прості, як правило, однотипні, елементи (комірки), що імітують принцип роботи нейронів мозку. Ці елементи називають нейронами. Кожен нейрон характеризується своїм поточним станом за аналогією з нервовими клітинами головного мозку, які можуть бути збуджені або загальмовані. У нього є група синапсів - односпрямованих вхідних зв'язків, з'єднаних з виходами інших нейронів, а також аксон – вихідний зв'язок даного нейрона, з якого сигнал (збудження або гальмування) надходить на синапси наступних нейронів.

Багатошарова нейронна мережа складається з нейронів, розташованих на різних шарах, причому, крім вхідного та вихідного шарів, є внутрішні (приховані) шари (рис.1.1). Таку нейронну мережу також називають *багатошаровим перцептроном*.

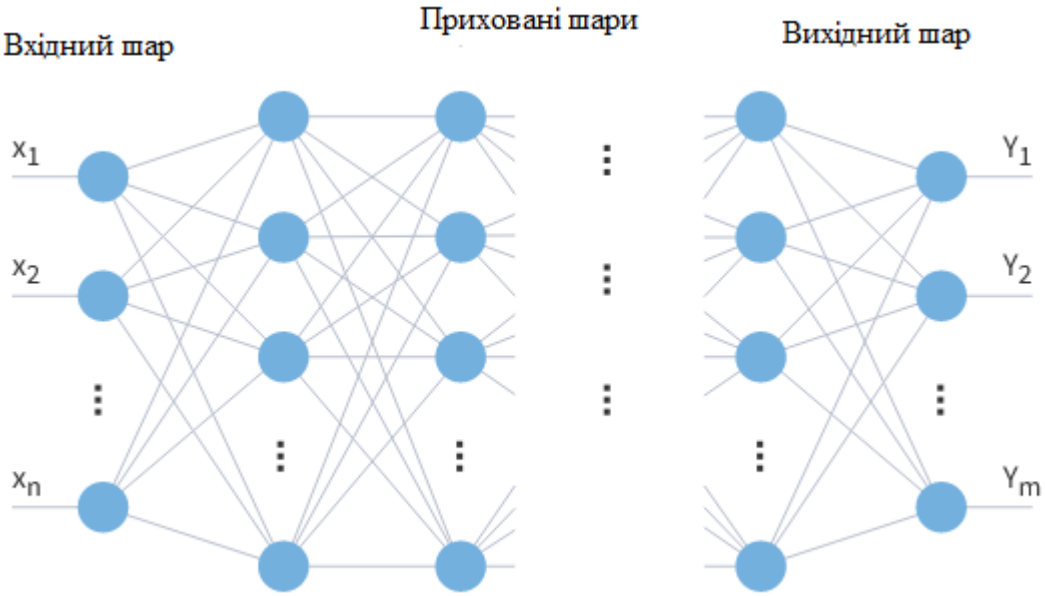


Рис.1.1. Багатошарова нейронна мережа

Кожен нейрон має певне значення вагових коефіцієнтів w_{ij} для всіх його зв'язків з попередніми на наступними шарами, а також кожний шар має поляризацію w_{i0}, на котру поступає одиничний сигнал.

На вхід системи надходить вектор сигналів $\langle x_j \rangle$, значення вагів змінюються в процесі навчання, котре наближує вихідні сигнали $\langle y_i \rangle$ до очікуваних сигналів $\langle d_i \rangle$. Мірою наближення є значення цільової функції $E(w_{ij})$. При використанні p навчаючих векторів $\langle \vec{x}, \vec{d} \rangle$ для навчання мережі з M вихідними нейронами, цільова функція визначається евклідовою метрикою у вигляді

$$E = \frac{1}{2}\sum_{k=1}^{p}\sum_{i=1}^{M}\left(y_i^{(k)} - d_i^{(k)}\right)^2 \tag{1.1}$$

Вихідні сигнали нейронів прихованого шару позначають $\nu_j, j = \overline{1,K}$, а вихідного шару $y_i, i = \overline{1,M}$.

В мережі з одним прихованим шаром, з врахування сигналу поляризації, вихідний сигнал $i-$ го нейрону прихованого шару описується функцією

$$\nu_i = f\left(\sum_{j=0}^{N} w_{ij}^{(1)} x_j\right) \tag{1.2}$$

При цьому для входу на наступному шарі $\nu_i = 1$. У вихідному шарі $k-$ ий нейрон видає сигнал

$$y_k = f\left(\sum_{i=0}^{K} w_{ki}^{(2)} \nu_i\right) = f\left(\sum_{i=0}^{K} w_{ki}^{(2)} \left(\sum_{j=0}^{N} w_{ij}^{(1)} x_j\right)\right) \tag{1.3}$$

Отже, на сигнали вихідного шару впливають ваги як зовнішнього, так і прихованого шарів.

Навчання проводиться в декілька етапів:

1) Системі пред'являється навчальна вибірка $\langle x_j \rangle$ та розраховуються значення сигналів відповідних нейронів мережі. Для заданого $\langle x_j \rangle$ визначаються значення

вихідних сигналів прихованого шару $\langle v_i \rangle$, а потім – значення сигналів вихідного шару $\langle y_i \rangle$ (якщо прихований шар - один) за формулами (1.2) і (1.3).

2) Методами оптимізації мінімізується значення цільової функції (1.1). Для неперервної цільової функції використовують метод градієнтного спуску, за яким уточнення вагів визначається формулою

$$\vec{w}(k+1) = \vec{w}(k) + \Delta\vec{w} \qquad (1.4)$$

де $\Delta\vec{w} = \eta p(\vec{w})$, η - швидкість навчання, $p(\vec{w})$ - напрям у багатовимірному просторі $\vec{w}$.

Для навчання внутрішніх шарів необхідно враховувати вплив цих шарів на вихідний сигнал мережі і похибку, яка виникає під цим впливом. Тому для навчання внутрішніх було запропоновано спеціальну стратегію, яка називається алгоритмом зворотного поширення помилки (backpropagation).

При навчанні ваги нейронів корегуються в бік мінімізації цільової функції за градієнтом

$$\nabla_j E = \frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial y_i} \cdot \frac{dy_i}{du_i} \cdot \frac{\partial u_i}{\partial w_{ij}} = (y_i - d_i) f'(u_i) \cdot x_j \qquad (1.5)$$

Формула (1.5) застосовується для навчання нейронів вихідного шару.

Для j-го нейрону попереднього шару (рис.1.2) отримуємо

$$\frac{\partial E}{\partial y_i} = \sum_{k=1}^{K^{n+1}} \frac{\partial E}{\partial y_k} \cdot \frac{dy_k}{du_k} \cdot \frac{\partial u_k}{\partial y_j} = \sum_{k=1}^{K^{n+1}} \frac{\partial E}{\partial y_k} \cdot f'(u_i) \cdot w_{jk}^{(n+1)} \qquad (1.6)$$

де сумування ведеться по всім К нейронам наступного шару $(n+1)$, пов’язаного з j-м нейроном поточного шару n.

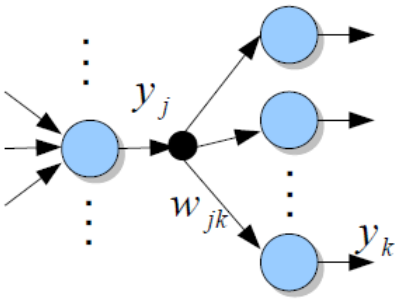


Рис.1.2. Зв'язок j-го нейрону попереднього n-го шару з К нейронами наступного шару $(n+1)$

Введемо позначення $\delta_j^{(n)} \stackrel{\text{def}}{=} \frac{\partial E}{\partial y_j} \cdot f'(u_j)$. Запишемо рекурентне співвідношення

$$\frac{\partial E}{\partial y_j^{(n)}} \cdot f'(u_j) = \left(\sum_{k=1}^{K^{n+1}} \delta_k^{(n+1)} w_{jk}^{(n+1)} \right) \cdot f'(u_j) = \delta_j^{(n)} \qquad (1.7)$$

При цьому для вихідного шару

$$\delta_j^{(N)} = (y_i^N - d_i) f'(u_i) \qquad (1.8)$$

Тоді для довільного шару компоненти градієнта розраховуються за формулою

$$\Delta w_{ij}^{(n)} = -\eta \delta_j^{(n)} y_i^{(n-1)} \qquad (1.8)$$

де для першого шару $y_i^{(1)} = x_i$.

Для позбавлення від ефекту осціляції рекомендується вводити коефіцієнти інерційності навчання

$$\Delta w_{ij}^{(n)}(t+1) = -\eta \delta_j^{(n)} y_i^{(n-1)} + a w_{ij}^{(n)}(t), \qquad |\alpha| < 1 \qquad (1.9)$$

або

$$\Delta w_{ij}^{(n)}(t+1) = -\eta \left(\mu w_{ij}^{(n)}(t) + (1-\mu) \delta_j^{(n)} y_i^{(n-1)} \right), \qquad |\mu| < 1 \qquad (1.10)$$

3) Після зміни вагів процедура подачі навчальної вибірки повторюється до тих пір, поки цільова функція не досягне заданого порогу значень ($E \leq \varepsilon$).

Ідеї побудови багатошарових нейронних мереж запропоновані відносно давно. У 1974 р. Галушкін А.І[3] і Дж. Вербос[4] незалежно один від одного запропонували алгоритм оберненого навчання. Все в середині 1980-х років галузь розробок нейронних мереж оцінювалась в $7 млрд.дол., а на початку 1990-х досягла $120 млрд.дол[5]. З тих пір системи, побудовані на нейронних мережах, нарощують свою присутність у нашому житті, нещодавно подолавши трильйонні рубежі.

При практичному застосуванні виділяють наступні етапи навчання нейронної мережі:

---

[3] Галушкин А. И. Синтез многослойных систем распознавания образов. — М.: Энергия, 1974.

[4] Werbos P. J. Beyond regression: New tools for prediction and analysis in the behavioral sciences. — Ph. D. thesis, Harvard University, Cambridge, MA, 1974.

[5] Aldrich C. Exploratory Analysis of Metallurgical Process Data with Neural Networks and Related Methods. - Elsevier, 2002. — 387 p. – p.23

- Збір, підготовка та нормалізація даних;
- Обґрунтування архітектури мережі;
- Експериментальний вибір характеристик мережі;
- Експериментальний вибір параметрів навчання;
- Навчання;
- Перевірка адекватності навчання (верифікація навченої мережі);
- Коригування параметрів, остаточне навчання;
- Документування отриманих результатів з метою подальшого використання.

На сучасних виробництвах системи з використанням нейронних мереж найчастіше застосовуються для управління якістю виробництва, управління технологічними процесами та відеоаналітики.

Зокрема, дослідники[6] із Чехії зазначають, що системи зі штучним інтелектом покращують продуктивність і ефективність промислових підприємств. При цьому використовуються не лише аналітичні і профілактичні, а також прогнозні можливості навчених систем, що дозволяє суттєво знижувати витрати на утримання активів. Впровадження новітніх методів революційно змінює методи обробки даних, мінімізує ризик людської помилки, забезпечує широкий спектр можливостей для підвищення ефективності виробництва та реалізації рішень Business Intelligence (BI).

Польські фахівці[7] загострюють увагу на труднощах при проектування систем реального часу із застосуванням штучного інтелекту. В практичній частині статті розглядаються можливості багатошарових нейронних мереж для динамічного управління тепловою станцією.

В Україні системи штучного інтелекту також впроваджуються не лише фінансовими компаніями, але й промисловцями. Один з лідерів українського

[6] Landryova, L., Sikora J., Wagnerová R. The Learning Path to Neural Network Industrial Application in Distributed Environments // Processes 2021, 9, 2247. https://doi.org/10.3390/pr9122247

[7] Patan K., Korbicz J. Application of Dynamic Neural Networks in an Industrial Plant // 4th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes 2000 (SAFEPROCESS 2000), Budapest, Hungary, 14-16 June 2000 // https://doi.org/10.1016/S1474-6670(17)37358-5

пивоварного ринку, компанія AB InBev Efes, застосовує штучний інтелект та нейромережі у своїй комплаєнс-системі тому, що це мінімізує ризик людської помилки під час оцінки ступеня ризику, а також суттєво підвищує швидкість виявлення ризиків, що дозволяє зберігати високий рівень безпеки компанії та її співробітників [8].

Широко застосовуються нейронні мережі у системах відеоспостереження та контролю. Класифікація об'єктів дозволяє нейромережам знаходити те, що потрібно, оцінювати ризики та загрози: аварія на технологічній лінії, застосування несправного інструменту, знаходження без спецодягу в промисловому приміщенні або на будівельному майданчику тощо.

Таким чином, на сьогодні системи штучного інтелекту широко впроваджуються у промисловості, виводячи підприємтсва на новий рівень конкурентоспроможності, розширюючи можливості аналізу, управління, планування та прогнозу.

1.2. Обґрунтування структури згортки отриманих дискретних квазігармонійних сигналів у характеристичні вектори для навчання нейронних мереж

У даній роботі пропонується система на нейронних мережах, що здатна розрізняти штатну роботу та декілька аварійних ситуацій із високою точністю. Сигнали для аналізу поступають від датчика частоти обертів, встановленого на працюючому двигуні.

В якості датчиків частоти обертання в системах автоматики застосовують *тахогенератори* - малопотужні електричні прибори постійного та змінного струму. Для перетворення частоти обертання електродвигунів у напругу застосовують тахометричні мости.

[8] https://mind.ua/ru/news/20233124-ab-inbev-efes-primenyaet-v-sisteme-komplaensa-iskusstvennyj-intellekt-i-nejroseti

У роботі застосовується тахогенератор постійного струму (рис.1.3). Принцип його роботи подібний до звичайного колекторного генератора постійного струму, із постійним збудженням, що здійснюється постійними магнітами статора. Оскільки ЕРС, що наводиться в обмотках ротора, прямо пропорційна швидкості зміни магнітного потоку в обмотках відповідно до закону Фарадея, то і напруга, що знімається з щіток колектора, є прямо пропорційною швидкості обертання ротора.

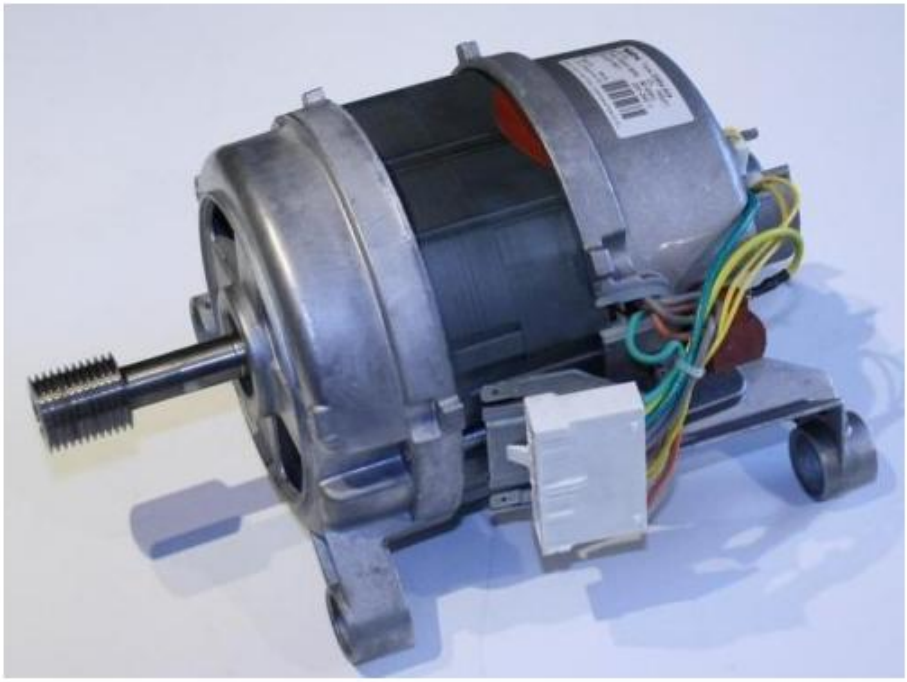


Рис.1.3. Електричний двигун, обладнаний тахогенератором постійного струму

Інформаційним сигналом тахогенератора постійного струму є електрична напруга. Сигнал має певний шум, що обумовлений, в основному, залежністю магнітного потоку підмагнічування від температури, перехідного електричного опору між щітками і колектором, зміни магнітного потоку підмагнічування постійного магніту статора з часом зміни зазору між ротором та статором.

Перевагою тахогенератору постійного струму є зручне формування вихідного сигналу і можливість визначати не тільки швидкість обертання ротора, а й напрямок його обертання (при зміні напряму обертання вихідний сигнал змінює полярність).

Сигнал від тахогенератора поступає на комп'ютер, обробляється і записується у файли даних (значення напруги) за установлений час.

Отриманий сигнал перед подачею на вхід нейронної мережі повинен згортатися. Концепція згортки побудована на моделюванні сигналу як композиції «гармоніка + білий шум»:

$$x(t) = A\cos(\omega t) + \eta_{rand}A \qquad (1.11)$$

де A – амплітуда, ω – циклічна частота, η_{rand} – випадковий коефіцієнт у заданому діапазоні $NSR_{min} \le \eta_{rand} \le NSR_{max}$; NSR_{min}, NSR_{max} – відповідно нижня та верхня границі відношення шуму до сигналу (Noise to Signal Ratio).

Для заданих меж NSR с випадковий коефіцієнт можна розрахувати за наступною формулою:

$$\eta_{rand} = (NSR_{max} - NSR_{min})R(0;1) + NSR_{min} \qquad (1.12)$$

де $R(0;1)$ – випадкове дійсне число в інтервалі $[0;1]$.

За допомогою формул (1.1) та (1.2) можна моделювати періодичні сигнали реальних механізмів (наприклад, обертання валу двигуна) або приладів (наприклад, струм чи напруга на ділянці ланцюга).

Вважатимемо сигнал *позитивним (нормальним)*, якщо відношення шуму до сигналу не перевищує 5%, тобто:

$$NSR_{min} = 0, \qquad NSR_{max} = 0{,}05; \qquad 0 \le \eta_{rand} \le 0{,}05 \qquad (1.13)$$

Сигнали, що моделюються формулами (1.11)-(1.13), позначатимемо літерою Р – як позитивні.

Ознаками неполадок (перешкод) у реальній системі можуть бути:

1) зростання рівня шуму у сигналі;

2) підвищення амплітуди сигналу;

3) зниження амплітуди сигналу;

4) підвищення частоти коливань;

5) зниження частоти коливань.

Отже, *неполадка* – це систематичне відхилення сигналу системи від норми. Усі неполадки практично є негативними сигналами, коли потрібна негайна реакція (відключення системи чи запуск системи управління).

У таблиці 1.1 представлені підходи до моделювання перерахованих п'яти неполадок.

Таблиця 1.1

Сигнали з неполадками та їх моделювання

Нотація	Опис неполадки	Модель
$N1$	Зростання рівня шуму	Шум з NSR від 10% до 20% Для моделювання використовуються формули (1) і (2) з $NSR_{min} = 0{,}1;\ \ NSR_{max} = 0{,}2$
$N2$	Зростання амплітуди	Амплітуда систематично перевищує норму в інтервалі від 5% до 20% Вводимо межі $A_{min} = 1{,}05A; A_{max} = 1{,}2A$ У формулі (1) замість постійної амплітуди A використовуємо змінну $A_{var} = (A_{max} - A_{min})R(0;1) + A_{min}$ (1.14)
$N3$	Зниження амплітуди	Амплітуда систематично нижча за норму в інтервалі від 5% до 20% Для моделювання використовуємо формулу (4) з $A_{min} = 0{,}8A;\ \ A_{max} = 0{,}95A$
$N4$	Зростання частоти	Частота систематично перевищує норму в інтервалі від 5% до 20% Вводимо межі $\omega_{min} = 1{,}05\omega; \omega_{max} = 1{,}2\omega$ У формулі (1) замість постійної частоти ω використовуємо змінну $\omega_{var} = (\omega_{max} - \omega_{min})R(0;1) + \omega_{min}$ (1.15)
$N5$	Зниження частоти	Частота систематично нижча за норму в інтервалі від 5% до 20% Для моделювання використовуємо формулу (5) з $\omega_{min} = 0{,}8\omega;\ \ \omega_{max} = 0{,}95\omega$

В даному розділі ми пропонуємо методику згортки для класифікації нормального сигналу P та перешкод N1-N5 за допомогою нейронних мереж з точністю не менше 98%.

Перепишемо сигнал (1) у вигляді:

$$x(t) = A\cos(\omega t) + n(t) \tag{1.16}$$

де $n(t)$ - це шум. Для гармонічної складової сигналу (1.6) без шуму можемо записати відомі співвідношення:

$$x_0(t) = A\cos(\omega t + \varphi), \qquad v_0(t) = x_0'(t) = -\omega A\sin(\omega t + \varphi) \tag{1.17}$$

$$a_0(t) = v_0'(t) = x_0''(t) = -\omega^2 A\cos(\omega t + \varphi)$$

Таким чином, амплітуди швидкості та прискорення зміни сигналу

$$v_{0\,max} = \omega A, \qquad a_{0\,max} = \omega^2 A \tag{1.18}$$

несуть в собі інформацію про частоту сигналу.

Середня потужність сигналу

$$\langle P_0\rangle = \frac{1}{T}\int_0^T x^2(t)dt = \frac{A^2}{T}\int_0^T \cos^2(\omega t + \varphi)\,dt = \tag{1.19}$$

$$= \frac{A^2}{2T}\int_0^T \big(1 + \cos\big(2(\omega t + \varphi)\big)\big)\,dt =$$

$$= \frac{A^2}{2T}\Big(t + \frac{1}{2\omega}\sin\big(2(\omega t + \varphi)\big)\Big)\Big|_0^T = \frac{A^2}{2}$$

пропорційна квадрату амплітуди.

У реальних системах отриманий від детектора сигнал є дискретним набором точок. $x(t_i), i = \overline{1,N}$. За цим набором можна визначити максимальне, мінімальне та середнє відхилення:

$$x_{max} = \max_i x(t_i), \qquad x_{min} = \min_i x(t_i), \qquad \langle x\rangle = \frac{1}{N}\sum_{i=1}^{N} x(t_i) \tag{1.20}$$

$$x_{max}\sim A, \qquad x_{min}\sim -A, \qquad \langle x\rangle\sim\langle n(t)\rangle$$

Отримуємо характеристики амплітуди та середнього рівня шуму (з огляду на те, що середня для гармонічної складової сигналу дорівнює нулю).

Швидкість зміни сигналу на дискретному наборі визначається формулою:

$$v(t_i) = \frac{x(t_i) - x(t_{i-1})}{\Delta t}, \qquad i = \overline{2, N} \tag{1.21}$$

де $\Delta t = t_i - t_{i-1} = const$ – крок за шкалою часу.

Для швидкості ми також можемо визначити максимальне, мінімальне та середнє значення::

$$v_{max} = \max_i v(t_i), \qquad v_{min} = \min_i v(t_i), \qquad \langle v \rangle = \frac{1}{N-1}\sum_{i=2}^{N} v(t_i) \tag{1.22}$$

$$v_{max} \sim \omega A, \qquad v_{min} \sim -\omega A, \qquad \langle v \rangle \sim \langle \frac{dn(t)}{dt} \rangle$$

Отримуємо характеристики частоти, амплітуди та швидкості зміни шуму.

Прискорення зміни сигналу на дискретному наборі:

$$a(t_i) = \frac{v(t_i) - v(t_{i-1})}{\Delta t}, \qquad i = \overline{3, N} \tag{1.23}$$

Для прискорення можемо записати:

$$a_{max} = \max_i a(t_i), \qquad a_{min} = \min_i a(t_i), \qquad \langle a \rangle = \frac{1}{N-2}\sum_{i=3}^{N} a(t_i) \tag{1.24}$$

$$a_{max} \sim \omega^2 A, \qquad a_{min} \sim -\omega^2 A, \qquad \langle a \rangle \sim \langle \frac{d^2 n(t)}{dt^2} \rangle$$

Отримуємо характеристики квадрата частоти, амплітуди та прискорення зміни шуму. За отриманими характеристиками, враховуючи: $x_{max} - x_{min} \sim 2A$, $v_{max} - v_{min} \sim 2\omega A$, $a_{max} - a_{min} \sim 2\omega^2 A$ для даного зразку можемо також розрахувати:

$$\langle \omega \rangle = \frac{v_{max} - v_{min}}{x_{max} - x_{min}}, \qquad \langle \omega^2 \rangle = \frac{a_{max} - a_{min}}{x_{max} - x_{min}}, \tag{1.25}$$

$$\langle P \rangle = \frac{(x_{max} - x_{min})^2}{8}$$

Додаємо показники варіації ознак.

Середні квадратичні відхилення

$$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x(t_i) - \langle x\rangle)^2}\,, \qquad \sigma_v = \sqrt{\frac{1}{N-2}\sum_{i=2}^{N}(v(t_i) - \langle v\rangle)^2}$$

$$\sigma_a = \sqrt{\frac{1}{N-3}\sum_{i=3}^{N}(a(t_i) - \langle a\rangle)^2} \qquad (1.26)$$

Лінійне відхилення за потужністю:

$$d_p = \left|\frac{1}{2N}\sum_{i=1}^{N} x^2(t_i) - \langle P\rangle\right|$$

Отже, для будь-якого квазігармонічного процесу, який описується дискретним сигналом $x(t_i)$ можна сформувати характеристичний вектор з 16 складових (1.20),(1.22),(1.24),(1.25),(1.26), які в сукупності описують амплітуду, частоту, потужність основного сигналу, характеристики шуму та показники варіації (табл.1.2).

Таблиця 1.2

Компоненти характеристичного 16-вектора
для дискретних квазігармонійних сигналів

№	Компонента	Формула розрахунку	Характеристикою чого є дана компонента
1	Максимальне відхилення	$x_{max} = \max_i x(t_i)$	$x_{max} \sim A$ Позитивне значення амплітуди
2	Мінімальне відхилення	$x_{min} = \min_i x(t_i)$	$x_{min} \sim -A$ Негативне значення амплітуди
3	Середнє відхилення	$\langle x\rangle = \frac{1}{N}\sum_{i=1}^{N} x(t_i)$	$\langle x\rangle \sim \langle n(t)\rangle$ Середній рівень шуму
4	Максимальна швидкість	$v_{max} = \max_i v(t_i)$	$v_{max} \sim \omega A$ Частота і амплітуда «зверху»
5	Мінімальна швидкість	$v_{min} = \min_i v(t_i)$	$v_{min} \sim -\omega A$ Частота і амплітуда «знизу»

Продовження таблиці 1.2

№	Компонента	Формула розрахунку	Характеристикою чого є дана компонента
6	Середня швидкість	$\langle v\rangle = \frac{1}{N-1}\sum_{i=2}^{N} v(t_i)$	$\langle v\rangle \sim \langle \frac{dn(t)}{dt}\rangle$ Середня швидкість зміни шуму
7	Максимальне прискорення	$a_{max} = \max_i a(t_i)$	$a_{max} \sim \omega^2 A$ Квадрат частоти і амплітуда «зверху»
8	Мінімальне прискорення	$a_{min} = \min_i a(t_i)$	$a_{min} \sim -\omega^2 A$ Квадрат частоти і амплітуда «знизу»
9	Середнє прискорення	$\langle a\rangle \sim \langle \frac{d^2n(t)}{dt^2}\rangle$	$\langle a\rangle \sim \langle \frac{d^2n(t)}{dt^2}\rangle$ Середнє прискорення зміни шуму
10	Середня частота	$\langle \omega\rangle = \frac{v_{max} - v_{min}}{x_{max} - x_{min}}$	Середня частота
11	Середня квадрату частоти	$\langle \omega^2\rangle = \frac{a_{max} - a_{min}}{x_{max} - x_{min}}$	Середня квадрату частоти
12	Середня потужність	$\langle P\rangle = \frac{(x_{max} - x_{min})^2}{8}$	Середня потужність
13	СКВ для відхилення	$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x(t_i) - \langle x\rangle)^2}$	Варіація відхилення
14	СКВ для швидкості	$\sigma_v = \sqrt{\frac{1}{N-2}\sum_{i=2}^{N}(v(t_i) - \langle v\rangle)^2}$	Варіація швидкості
15	СКВ для прискорення	$\sigma_a = \sqrt{\frac{1}{N-3}\sum_{i=3}^{N}(a(t_i) - \langle a\rangle)^2}$	Варіація прискорення
16	Лінійне відхилення для потужності	$d_p = \left\lvert \frac{1}{2N}\sum_{i=1}^{N} x^2(t_i) - \langle P\rangle \right\rvert$	Варіація потужності

Таким чином, ми пропонуємо перетворення будь-якого дискретного квазігармонічного сигналу $x(t_i), i = \overline{1,N}$ - довільної тривалості - в набір із 16 характеристик, які повністю його описують. Тим самим вектору сигналу розмірністю N зіставляється характеристичний вектор розмірності 16.

Переваги даного підходу перетворення сигналу для класифікації за допомогою нейромереж:

1) незалежність від тривалості сигналу N. В одній вибірці можуть знаходитися сигнали різної тривалості, але тривалість не впливає на зміст компонент 16-вектора, що застосовуються для навчання нейромереж;

2) незалежність від фази сигналу, що надійшов. В одній вибірці можуть бути квазігармонічні сигнали з різними фазами, але фаза не впливає на зміст компонент 16-вектора;

3) значне скорочення розмірності вхідних даних (згортка N-компонентного вектора довільної розмірності в 16-компонентний вектор).

1.3. Послідовність навчання та верифікації нейронних мереж

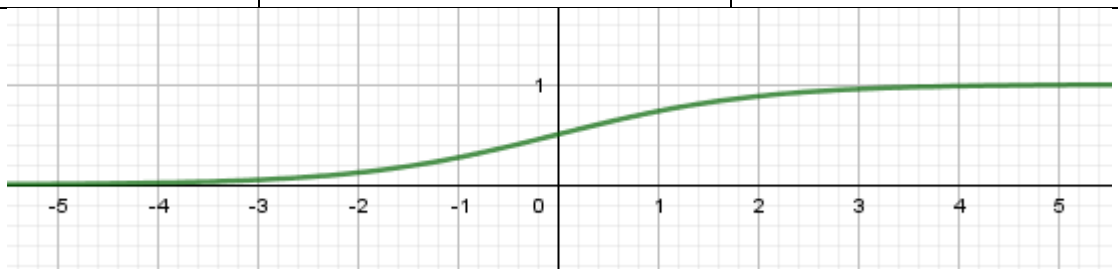
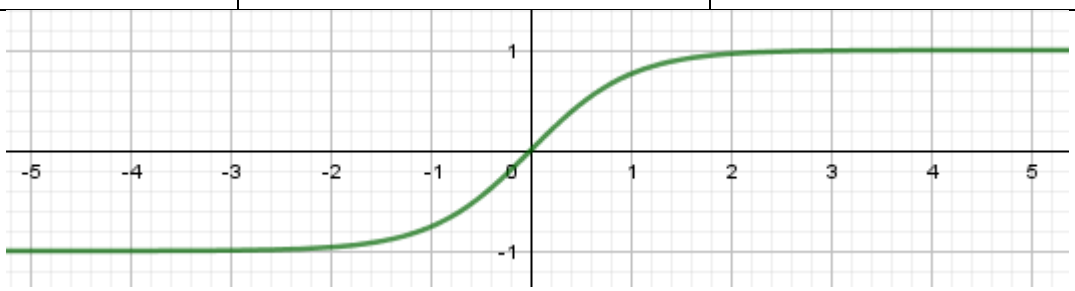
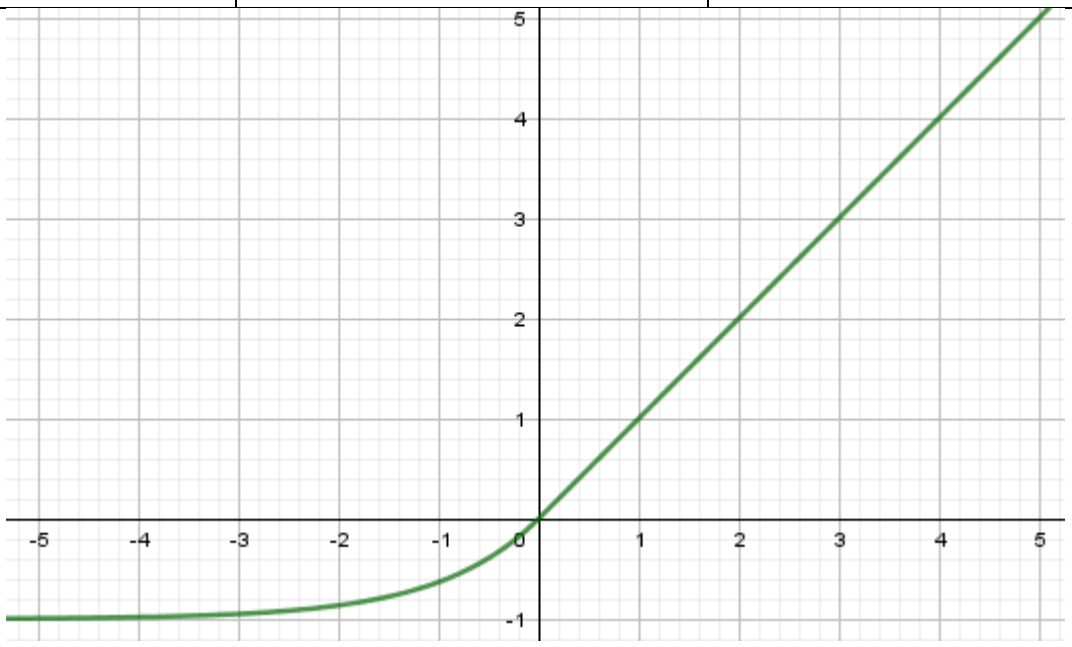
Характеристичні 16-вектори, що подаються на вхід нейронних мереж для навчання, мають бути нормовані.

Завдяки особливостям функцій активації нейронів, *вибір еталонного зразка та нормування на нього* є обов'язковим етапом підготовки зразків до навчання нейронних мереж. Розглянемо це питання детальніше.

Під час навчання нейронних мереж на сигнали реагують нейрони із заданою функцією активації. У розробленому програмному пакеті використовуються три функції: сигмоїда, гіперболічний тангенс та ELU (табл.1.3).

Таблиця 1.3

Функції активації нейромереж у розробленому програмному пакеті

	Назва	Формула	Область значень
1	Сигмоїда	$\sigma(x) = \frac{1}{1+e^{-x}}$	$(0;1)$
			
2	Гіперболічний тангенс	$th(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	$(-1;1)$
			
3	ELU (Exponential Linear Unit)	$f(\alpha, x) = \begin{cases} \alpha(e^{x}-1), & x < 0 \\ x, & x \geq 0 \end{cases}$	$(-\alpha;+\infty)$
			

Як видно із графіка, сигмоїда практично не розрізняє вхідні сигнали більше 4 за абсолютною величиною:

$$\sigma(5) \approx 0{,}9933, \qquad \sigma(6) \approx 0{,}9975$$

В той же час, сигнали в інтервалі (-1; 1), навіть при відстані на порядок менше, розрізняються на порядок краще:

$$\sigma(0{,}2) \approx 0{,}5498, \qquad \sigma(0{,}3) \approx 0{,}5744$$

Аналогічно, на гіперболічний тангенс не варто подавати вхідні сигнали, що перевищують 2 за абсолютної величиною. ELU, здавалося б, позбавлена цього недоліку; проте неконтрольоване зростання відгуку нейрона зрештою призводить до того, що ваги зв'язків між нейронами прямують до нескінченності.

Таким чином, робимо висновок: вхідні дані необхідно нормувати.

В даному дослідженні для нормування всієї сукупності 16-векторів (для $P, N1 - N5$ сигналів) використовується 16-вектор еталонного сигналу.

Для зашумленого гармонічного сигналу (1.1) еталонним є гармонічний сигнал без шуму:

$$x(t) = A\cos(\omega t) \tag{1.27}$$

Розрахунок характеристичного 16-вектору для еталонного зразка має наступні компоненти

$$\begin{array}{ccc} x_{\max et}, & x_{\min et}, & \langle x\rangle_{et} \\ v_{\max et}, & v_{\min et}, & \langle v\rangle_{et} \\ a_{\max et}, & a_{\min et}, & \langle a\rangle_{et} \\ \langle\omega\rangle_{et}, & \langle\omega^2\rangle_{et}, & \langle P\rangle_{et} \\ \sigma_{x\,et}, & \sigma_{v\,et}, & \sigma_{a\,et} \\ & d_{P\,et} & \end{array}$$

Кожна з компонент 16-векторів інших сигналів ділиться на відповідне еталонне значення, чим забезпечується її попадання в інтервал $[-a; a] \subset [-2; 2]$ при досліджуваних величинах відхилень (до 20% у той чи інший бік).

В Додатку А представлена зведена таблиця для нормування 16-векторів.

Розмірність характеристичних векторів $N = 16$ визначає розмірність вхідного шару нейронних мереж ($S -$ шар): на вході повинно бути kN нейронів. Тобто, розмірність має бути кратною 16. Характеристичні вектори можна подавати послідовно на нейрони входу декілька разів.

Кількість прихованих шарів ($A -$ шари) і число нейронів в них є довільним. В процесі експериментів можна знайти найбільш доцільну топологію з точки зору компромісу між швидкістю та якістю навчання. Складна топологія знижує швидкість, але в цілому підвищує якість, і навпаки.

З метою досягнення високої точності та надійності, необхідної для систем реального часу, кожна нейронна мережа проектується як класифікатор двох подій; наприклад, має розрізняти штатну роботу P від неполадки N1 (див.табл.1.1), або неполадку N2 від неполадки N5 тощо. Відповідно, на виході нейронної мережі ($R -$ шар) повинно бути $M = 2$ нейрони.

Нейросітка навчається в стохастичному режимі: всі доступні зразки подаються на $S -$ шар у рандомному порядку протягом епохи. *Епоха* – це одна ітерація у процесі навчання, під час якої нейронній мережі пред'являються всі зразки із навчальної множини. Кількість епох задається користувачем. Для управління процесом зворотного розповсюдження помилки використовуються два параметри: швидкість навчання η і момент α (див. формулу (1.9.)).

Візуально, успіх або невдача при навчанні можуть бути оцінені за графіком залежності цільової функції $E(n)$ (формула (1.1)) від номеру епохи. Якщо навчання проходить успішно, значення $E(n)$ прямує до 0 (рис.1.4). За цією оцінкою можуть відбраковуватися явно не навчені мережі.

Після вдалого за оцінкою $E(n)$ навчання кожна нейронна мережа повинна проходити процедуру верифікації. Для цього використовують частину отриманих зразків (30-50%), котрих нейронна мережа «не бачила» під час навчання. Якісно навчена мережа має класифікувати ці – нові для себе – сигнали із заданою точністю. Як правило, досягти точності 90% досить

нескладно, але далі підвищення на кожен процент за рахунок експериментального підбору топології потребує значного часу.

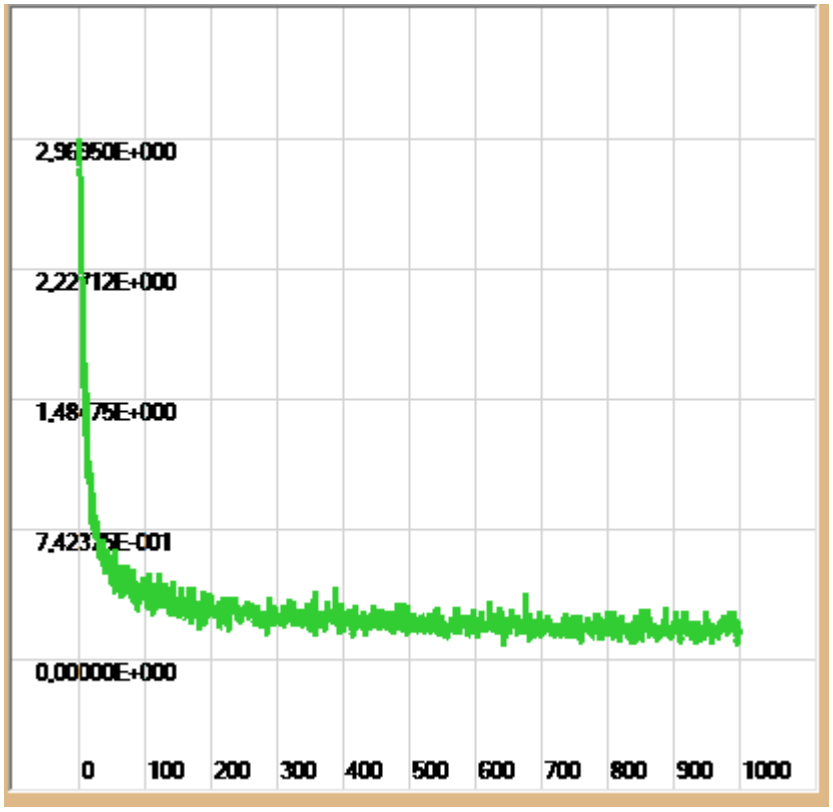


Рис.1.4. Динаміка змін цільової функції $E(n)$ в залежності від номеру епохи n при успішному навчанні

Точність реакції навченої нейронної мережі на позитивні та негативні зразки оцінюється за наступною методикою.

Параметри оцінки:

- стан позитивний (P) - кількість реальних позитивних зразків
- стан негативний (N) - кількість реальних негативних зразків
- дійсно позитивний (TP) - результат тесту, який правильно вказує на позитивний стан
- дійсно негативний (TN) - результат тесту, який правильно вказує на негативний стан
- помилково позитивний (FP) - результат тесту, який помилково вказує на позитивний стан

– помилково негативний (FN) - результат тесту, який помилково вказує на негативний стан.

За цими параметрами розраховуються 4 показники якості проведеного навчання (табл.1.3).

Таблиця 1.3

Процедура верифікації - показники якості проведеного навчання

Показник	Опис
$TPR = \frac{TP}{P} = \frac{TP}{TP + FN}$	Чутливість Відношення дійсно позитивних результатів до загальної кількості позитивних зразків
$TNR = \frac{TN}{N} = \frac{TN}{TN + FP}$	Селективність Відношення дійсно негативних результатів до загальної кількості негативних зразків
$ACC = \frac{TP + TN}{P + N}$	Точність Відношення суми дійсно розпізнаних зразків до загальної кількості зразків
$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{P \cdot N \cdot (TP + FP)(TN + FN)}}$	Коефіцієнт Метьюза Коефіцієнт кореляції для бінарних систем класифікації

Для кожної пари подій із множини штатної роботи та неполадок $\{P, N_1, N_2, N_3, N_4, N_5\}$ відбираються найбільш вдало навчені нейронні мережі, на котрих будується метаструктура, яка за комбінаторним принципом здатна розрізняти всі події, визначені у множині.

1.4. Принципи побудови метаструктури для сумісної роботи декількох навчених нейронних мереж

Припустимо, що нам необхідно розрізняти два види сигналів: штатну роботу (P) та неполадку (N_1). Для цього достатньо навчити одну нейромережу розпізнавати цю пару (табл.1.4).

Таблица 1.4

Нейромережа для розпізнавання двох класів сигналів: P i N_1

	P	N_1
P		✓
N_1		

Нехай u – нижня межа позитивного сигналу, d – верхня межа негативного сигналу; S - вхідний сигнал, P - каскад розпізнав сигнал S як штатну роботу, N_1 - каскад розпізнав сигнал S як неполадку 1, U - каскад не розпізнав сигнал.

Блок-схема найпростішого каскаду для розпізнавання двох сигналів представлена на рис.1.5.

Для того, щоб надійно розрізняти три види сигналів: штатну роботу (P) і дві різні неполадки (N_1 і N_2), нам знадобляться вже три нейромережі (табл.1.5).

Таблиця 1.5

Нейромережі для розпізнавання трьох класів сигналів: P, N_1 i N_2

	P	N_1	N_2
P		✓	✓
N_1			✓
N_2			

Блок-схема каскаду для розпізнавання трьох сигналів представлена на рис.1.6.

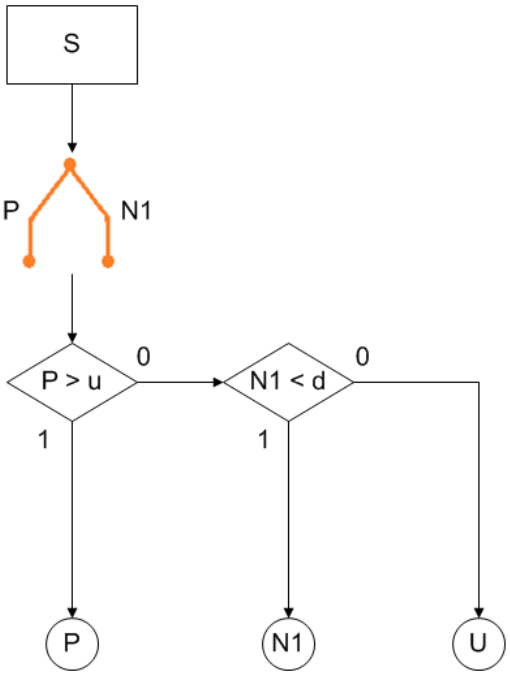


Рис.1.5. Блок-схема найпростішого каскаду для розпізнавання двох класів сигналів (одна нейромережа)

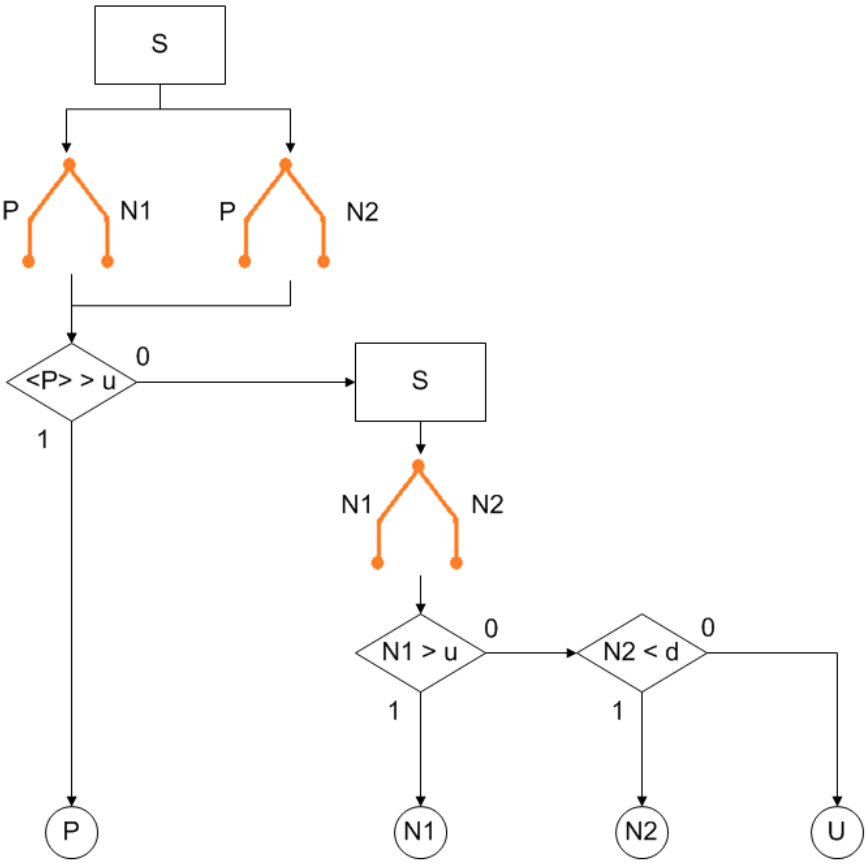


Рис.1.6. Блок-схема каскаду для розпізнавання трьох класів сигналів (три нейромережі), середня $\langle P \rangle = \frac{1}{k}\sum_{i=1}^{k} P_i$

Узагальнимо задачу для k неполадок (табл.1.6)

Таблиця 1.6

Нейромережі для розпізнавання $k+1$ класів сигналів $P, N_1, N_2, \ldots N_k$

	P	N_1	N_2	…	N_k
P		✓	✓	✓	✓
N_1			✓	✓	✓
N_2				✓	✓
…					✓
N_k					

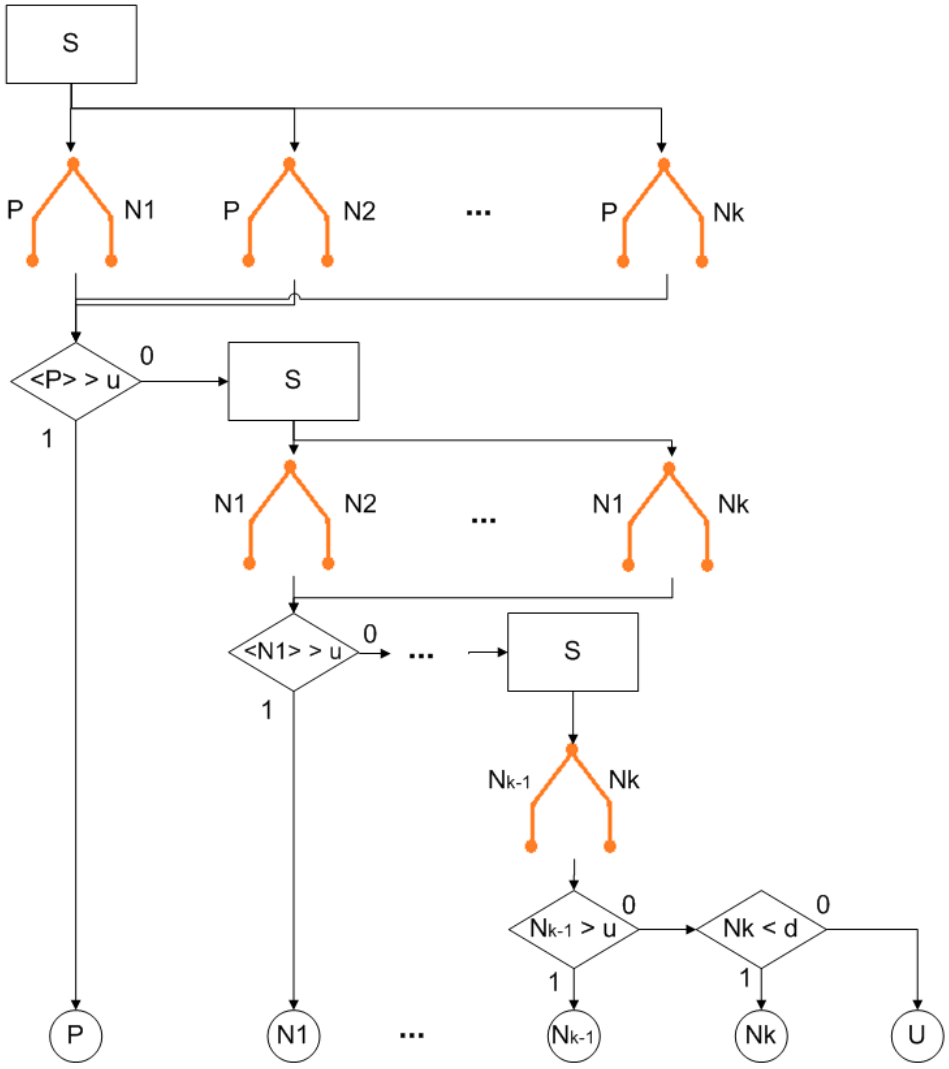


Рис.1.7. Блок-схема каскаду для розпізнавання $k+1$ класів сигналів

Нам знадобиться

$$m = \frac{k(k+1)}{2} \tag{1.28}$$

нейронних мереж, навчених розпізнавати пари сигналів з високою точністю.

Блок-схема каскаду для розпізнавання $k+1$ класів сигналів представлена на рис.1.7. Середні величини, за якими проходить «голосування» мереж, дорівнюють

$$\langle P \rangle = \frac{1}{k}\sum_{i=1}^{k} P_i\,, \qquad \langle N_j \rangle = \frac{1}{k-j}\sum_{i=1}^{k-j} N_{ji}\,, j = \overline{1, k-2} \tag{1.29}$$

Запропонована архітектура каскаду, що об'єднує декілька нейромереж , навчених розпізнавати пари класів сигналів з високою точністю, гарантує високу якість множинного розпізнавання. Додатковою перевагою цієї архітектури є можливість верифікації отриманої системи.

На нашу думку, точність порядку 95-99% і надійність пропонованого множинного каскадного класифікатора має зацікавити фахівців-практиків, що планують впровадження систем штучного інтелекту на виробництві.

1.5. Постановка задачі

Метою дослідження є розробка архітектури та програмна реалізація системи штучного інтелекту на основі багатошарових нейронних мереж із застосуванням згортки вхідних сигналів та метаструктури (каскаду) із навчених нейромереж, що здатна розрізняти штатну роботу та декілька передаварійних ситуацій із високою точністю.

Таким чином, побудований каскад із навчених нейронних мереж здатен прогнозувати поведінку реальної системи (електричного двигуна) на основі даних, що надходять у реальному часі (часових рядів). Тому пакет програмних рішень отримав назву ***TSForecast*** (прогнозування на основі часових рядів).

В системі здійснюються наступні операції:

- ручна і напівавтоматична розмітка часових рядів.
- створення нейромереж (ансамблів нейромереж) для прогнозування поведінки об'єкта. Доступні різні види мереж, функції активації вузлів, різноманітна архітектура та зв'язки між вузлами в одній мережі, а також між окремими мережами в ансамблі.
- навчання мереж з різними комбінаціями вхідних даних та режимів (розбивка на епохи, серії, мікропакети; параметри зворотного зв'язку тощо).
- верифікація мереж з отриманням звіту зі статистикою (TP,TN,FP,FN, похідні коефіцієнти, що характеризують чутливість, селективність, точність, та надійність навченої мережі).

Основні етапи роботи з пакетом TSForecast представлені на рис. 1.8.

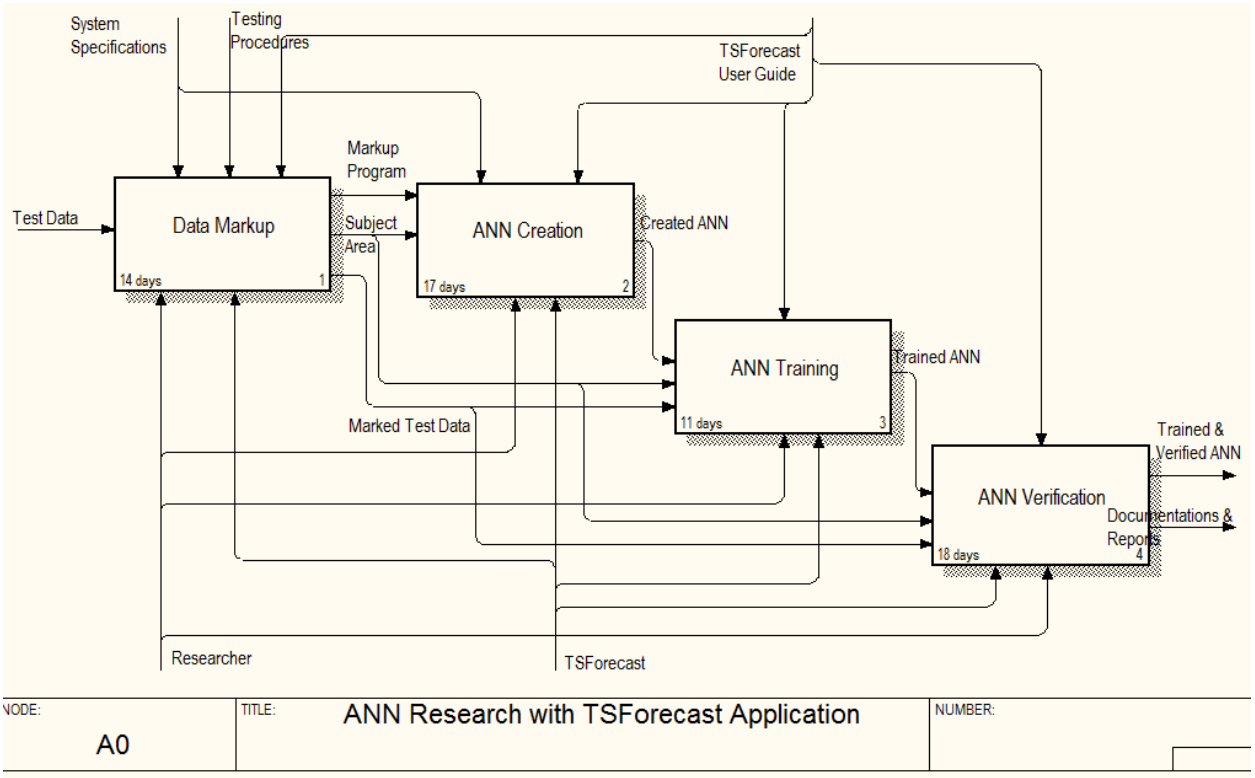


Рис.1.8. Чотири етапи досліджень з використанням пакету TSForecast на основі багатошарових нейронних мереж

На вході в процес (стрілка зліва) дані випробувань та/або тестів системи, що вивчається (тимчасові ряди).

Управління (правила, регламенти – стрілки зверху) представлене специфікаціями системи, що вивчається, протоколами випробувань та інструкціями користувача TSForecast.

Механізмами виконання програми (стрілки знизу) є дослідник і сама програма TSForecast.

Результатами проведеного дослідження (стрілки праворуч) будуть навчена та верифікована нейромережа, пакет документації та звітів.

На кожному з етапів виникають проміжні дані та матеріали, які необхідні для переходу на наступний рівень.

Наприклад, процес розмітки даних супроводжується створенням опису предметної області, програми розмітки – документації, необхідної для розробки нейромереж; також, на виході з цього етапу – розмічені дані випробувань, які будуть використовуватися під час навчання та верифікації.

Специфікації досліджуваної системи використовуються лише на етапах розмітки даних та створення нейромережі, опис процедур випробувань – лише на етапі розмітки. Інструкція користувача TSForecast необхідна кожному етапі, оскільки дає можливість повністю використовувати потенціал пропонованої системи штучного інтелекту.

Вимоги до технології розробки:

- IDE MS Visual Studio Community Edition 2017 або вище;
- мова C#.net 4.5 (можливість роботи на Windows 7x64 та вище).

Апаратні вимоги до користувачів:

- ПК що підтримуюсь Windows 7x64 та вище;
- обов’язкове використання GPU;
- рекомендоване використання багатоядерного CPU для ефективної паралельної обробки даних.

РОЗДІЛ 2. ПРОЕКТНІ РІШЕННЯ

2.1. Автоматизація згортки сигналів у 16-вектори, підготовка зразків для навчання

Декомпозиція Етапу 1 - підготовки даних для навчання нейромережі - подана на рис.2.1.

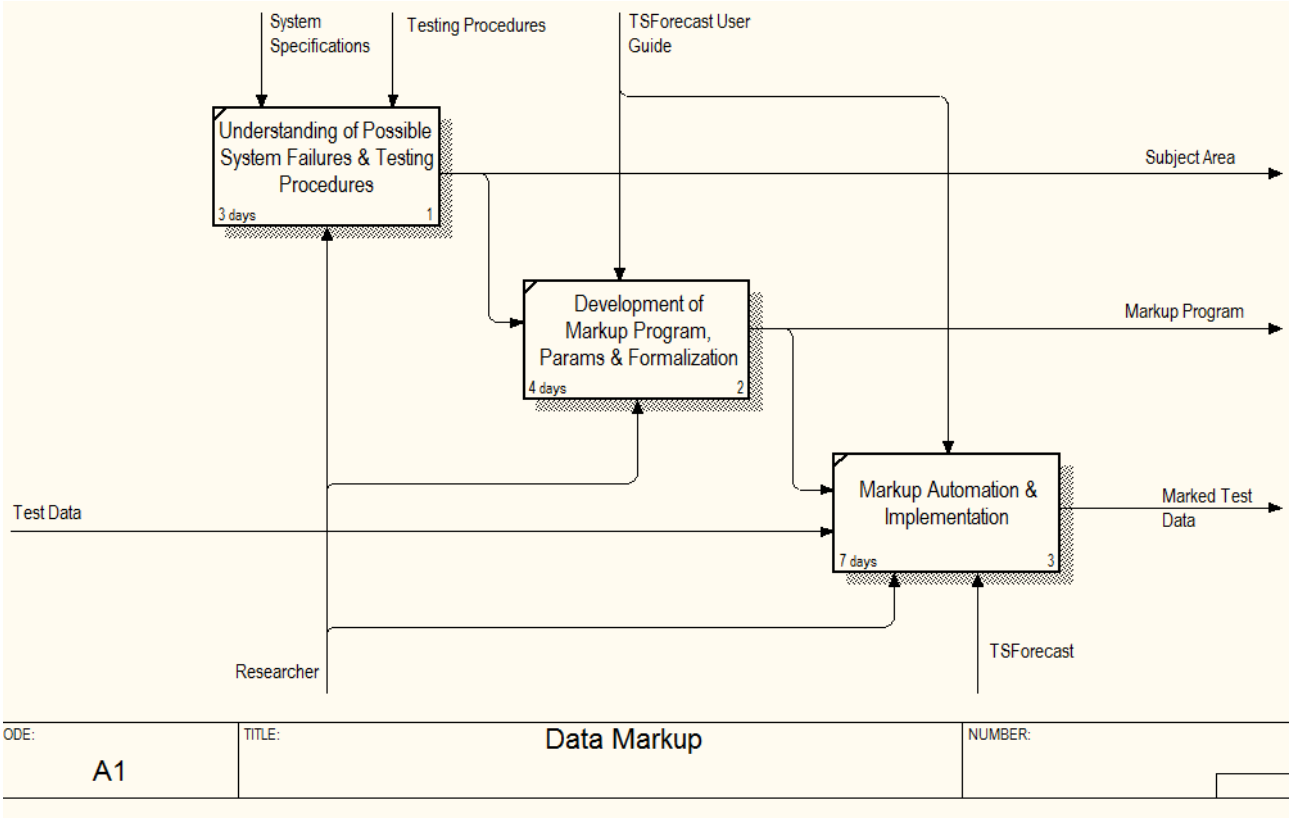


Рис.2.1. Декомпозиція Етапу 1: підготовка та розмітка даних

Предметна область для даного дослідження була детально розглянута у розділі 1.2. Позитивні сигнали виду (1.1) та негативні сигнали з різними неполадками, описаними в табл.1.1 (див. вище), були отримані за допомогою детекторів, встановлених на двигуні.

Запис сигналів – це файл з одним вектором виду $X(t)$, де кожен вимір X_i отримано через певний проміжок часу Δt (рис.2.2).

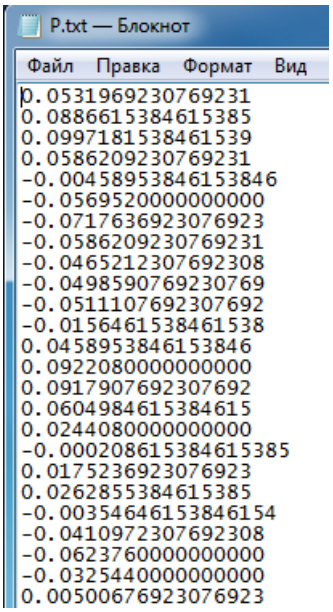


Рис.2.2. Тестовий файл P.txt з фрагментом отриманого ряду даних

Для візуалізації даних та їх первинної обробки в пакеті TSForecast було розроблено Модуль 1 «Візуалізація та маркування даних».

На рис.2.3 показано графічне зображення вектора даних із зразка P.txt.

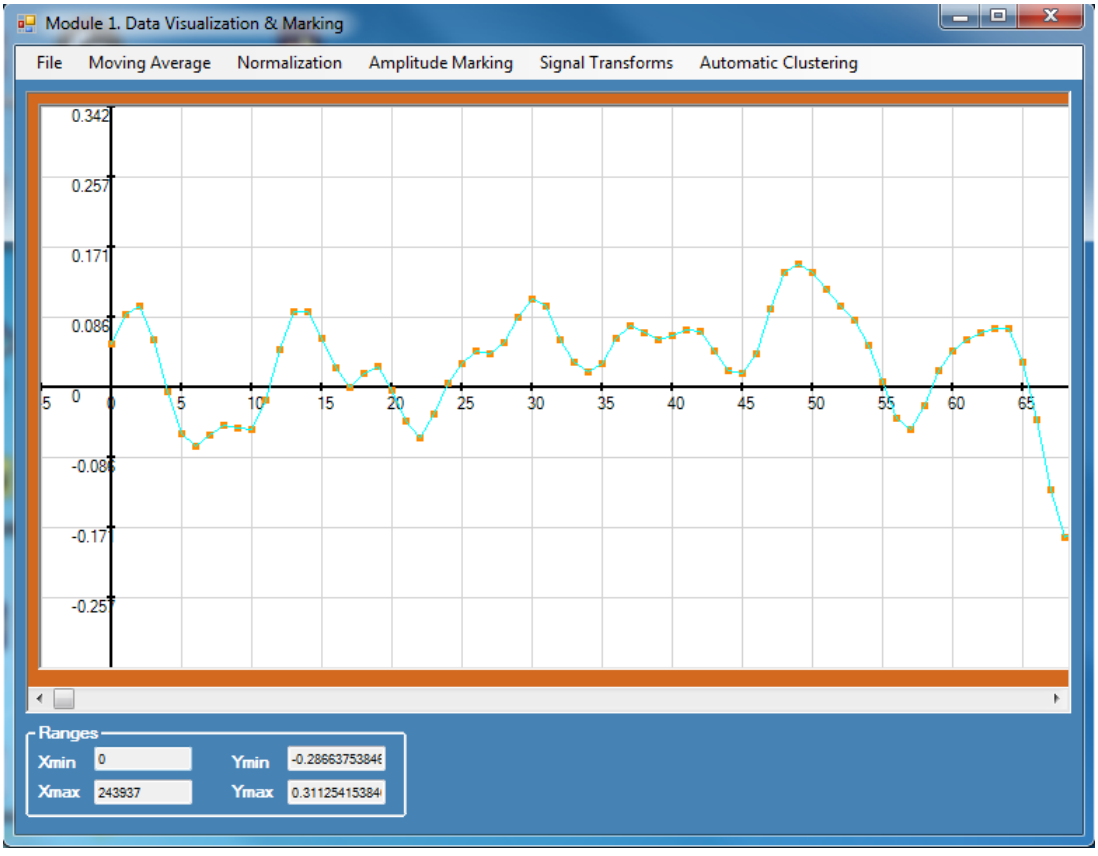


Рис.2.3. Графічне зображення вектора даних, отриманих від детектора, в Модулі 1 пакету TSForecast

Зразки отримуються протягом значного часу. Так в файлі P.txt 243937 точок. Ми можемо отримати з цього файлу певну кількість менших фрагментів. Розглянемо відповідне програмне рішення.

Нехай $T = 1000$ – довжина одного фрагмента, $h = 50$ – зсув для нової позиції кожного нового фрагмента, $N = 150$ - необхідна кількість фрагментів.

Алгоритм 2.1

Крок 1. Задати величини T, h, N

Крок 2. Завантажити файл з даними, знайти кількість t вимірів X_i у файлі, $i = \overline{1, t}$

Крок 3. Перевірити, чи можна отримати необхідну кількість фрагментів при заданих умовах

$$if(h \cdot N + T \leq t)$$

Якщо ні – крок 6 (кінець роботи).

Крок 4. Обрати папку для запису фрагментів.

Крок 5. У циклі $k = \overline{1, N}$

утворити множину точок Y в інтервалі $[hk; hk + T]$

записати вектор Y в файл в заданій папці

Крок 6. Кінець роботи

Таким чином, ми отримуємо необхідну кількість фрагментів із вхідними даними, що необхідні для роботи з нейронною мережею. Відповідний скрипт на мові MATLAB подано у Додатку Б.

Тестування Етапу 1 – нарізки даних та їх згортки у 16-вектори – проведено у розділі 3.1 даної роботи.

Зупинимося також на деяких додаткових можливостях розробленого на C# Модуля 1 «Візуалізація та маркування» пакету TSForecast. Для початку роботи, запускаємо на виконання TSF_Module1.exe (рис.2.4).

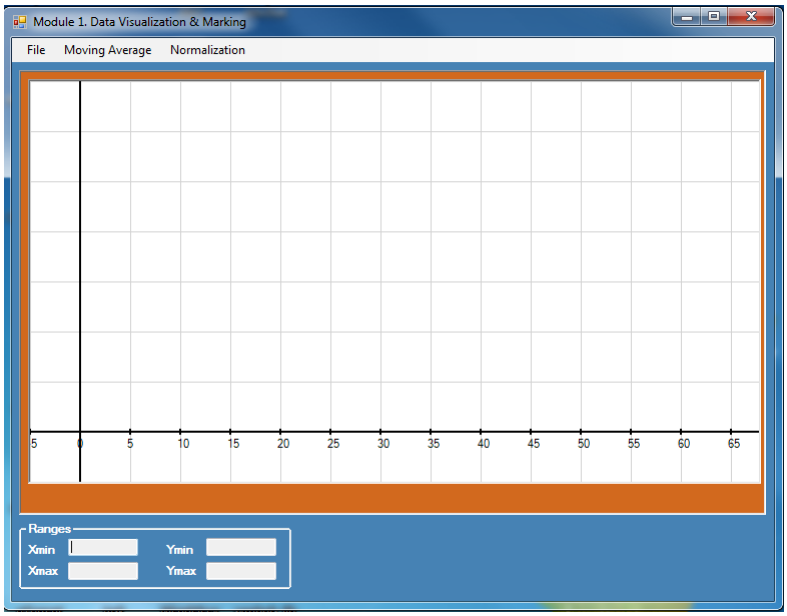


Рис. 2.4. Головне вікно Модуля 1 «Візуалізація та маркування»

Опції меню Модуля 1 показані на рис.2.5.

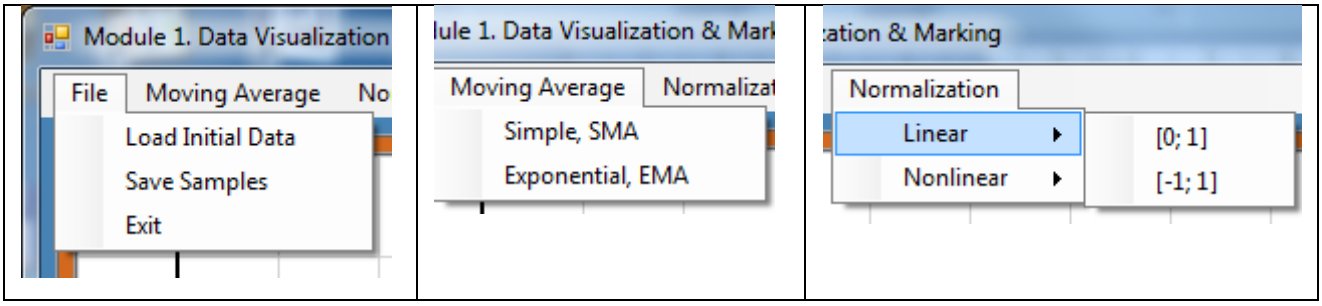


Рис.2.5. Опції меню Модуля 1

Для згладжування даних пропонуються два методи.

Ковзна середня (Simple Moving Average, SMA)

n – кількість спостережень

m – вікно згладжування, $m < n$

x_i - значення і-го спостереження, $i = \overline{0, n-1}$

Тоді згладжене значення сигналу дорівнює

$$x'_i = \frac{x_{i-m} + \cdots + x_i}{m}, i = \overline{m, n-1-m} \qquad (2.1)$$

Для решти i вікно згладжування поступово зростає на лівому кінці від 1 до m

Розрахунки розпаралелені.

Другий метод згладжування

Експоненційне згладжування (Exponential Moving Average, EMA)

α – коефіцієнт згладжування, $0 < \alpha < 1$

Може бути також виражений через вікно згладжування $m < n$: $\alpha = \frac{2}{m+1}$

n – кількість спостережень

x_i - значення і-го спостереження, $i = \overline{0, n-1}$

Тоді згладжене значення сигналу дорівнює

$$x'_i = \begin{cases} x_i, i = 0 \\ (1-\alpha)x'_{i-1} + \alpha x_i \end{cases} \quad (2.2)$$

Розрахунки не розпаралелюються.

Також, у Модулі 1 можна нормалізувати дані.

Лінійна нормалізація у відрізок $[0; 1]$ здійснюється за формулою

$$x'_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}, \qquad i = \overline{1, n} \quad (2.3)$$

де x_{min} – мінімальне значення сигналу у заданій множині вимірювань, x_{max} - максимальне значення сигналу, x_i - поточне значення сигналу, x'_i – нормалізоване значення сигналу.

Натомість, лінійна нормалізація у відрізок $[-1; 1]$ має вигляд

$$x'_i = 2\frac{x_i - x_{min}}{x_{max} - x_{min}} - 1, \qquad i = \overline{1, n} \quad (2.4)$$

Нелінійну нормалізацію можна здійснювати багатьма методами, зокрема, за допомогою розглянутих вище сігмоїди або гіперболічного тангенсу.

Наприклад, нелінійна нормалізація у відрізки $[0; 1]$ та $[-1; 1]$ має вигляд

$$x'_i = \frac{1}{e^{-a(x_i - x_C)}}, \qquad i = \overline{1, n} \quad (2.5)$$

$$x'_i = \frac{e^{a(x_i - x_C)} - 1}{e^{a(x_i - x_C)} + 1} \quad (2.6)$$

де $x_C = \frac{x_{min} + x_{max}}{2}$, параметр a задає ступінь нелінійності змінної x'_i.

Склад проекту TSF_Module1 представлено на рис.2.5.

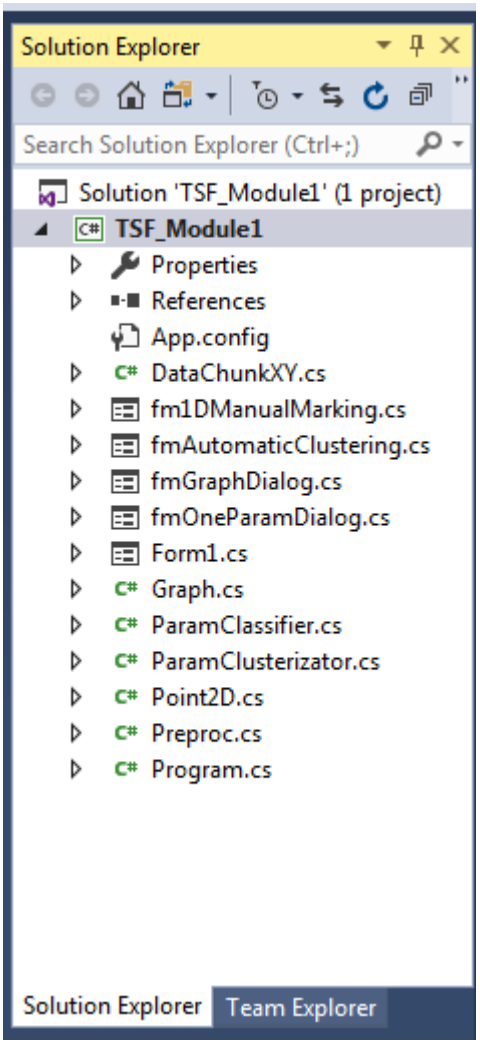


Рис.2.5. Склад проекту TSF_Module1

Для виконання поставлених завдань в проекті розроблено 6 класів-типів користувача та 5 класів-форм.

Опис основних класів, їх змінних та методів подано у табл.2.1.

Таблиця 2.1

Основні класи користувача проекту TSF_Module1

Клас	Опис, призначення
Класи-форми, що наслідують Form.Windows	

Form1.cs	Головна форма
	Змінні
	`Graphics g;`
	`Bitmap bmp;`
	`Graph graph;`
	`DataChunkXY initial_data;`
	`int step;`
	призначені для обробки графічного зображення
	Методи
	`public Form1() – конструктор`
	`private void PaintAll() – візуалізація даних`
	`private void openToolStripMenuItem_Click(object sender, EventArgs e) – завантаження файлу з даними`
	`private void Form1_Resize(object sender, EventArgs e) – подія – максимізація вікна`
	`private void simpleToolStripMenuItem_Click(object sender, EventArgs e) – згладжування SMA`
	`private void exponentialToolStripMenuItem_Click(object sender, EventArgs e) – згладжування EMA`
	`private void lhalfToolStripMenuItem_Click(object sender, EventArgs e) – лінійна нормалізація в [0;1]`
	`private void lfullToolStripMenuItem_Click(object sender, EventArgs e) - лінійна нормалізація в [-1;1]`

fmOneParamDialog.cs	Форма діалогу із запитом параметру
	Змінні
	`public bool ok; призначена для сигналу в головну форму про успішну операцію`
	Методи
	`fmOneParamDialog() – конструктор`
	`private void button1_Click(object sender, EventArgs e) – подія, натиснення кнопки OK`

fmGraphDialog.cs	Форма із візуалізацією результатів перетворень
	Змінні
	`Graphics g;`
	`Bitmap bmp;`
	`Graph graph;`
	`DataChunkXY data_to_paint;`
	`int step;`
	призначені для обробки графічного зображення
	Методи
	`fmGraphDialog(DataChunkXY datashow) – конструктор`
	`private void PaintAll() – візуалізація даних`
	`private void button1_Click(object sender, EventArgs e) – збереження отриманих результатів`

Продовження таблиці 2.1

Класи-типи користувача	

Graph.cs	Візуалізація даних
	Змінні
	`private int top, left, width, height;`
	`private Color bcg_color, grid_color, line_color;`
	`private double Xmin, Xmax, Ymin, Ymax;`
	`private double unitX, unitY;`
	`private int grid_step, x0, y0;`
	`private int m;`
	описують параметри зображення
	Методи
	`public Graph(int _top, int _left, int _width, int _height)` – конструктор
	`public void DrawGrid(Form form, Graphics g)` – побудова сітки
	`public void SetUnits(double xmin, double xmax, double ymin, double ymax)` – розрахунок масштабуючих множників
	`public void ArrangeY(Form form, Graphics g)` – зображення та надписи осі У
	`public void DrawPoint(Graphics g, Point2D p, int pwidth, SolidBrush point_brush)` – зображення точки
	`public void DrawLine(Graphics g, Point2D p1, Point2D p2, Pen lpen)` – зображення лінії

Point2D.cs	Точка на площині
	Властивості
	`public double X { set; get; }`
	`public double Y { set; get; }`
	Методи
	`public Point2D (double _X,double _Y)` – конструктор

Preproc.cs	Статичні методи математичної обробки `public static void SMA(DataChunkXY datain, int period, ref DataChunkXY dataout)` - Simple Moving Average `public static void EMA(DataChunkXY datain, int period, ref DataChunkXY dataout)` - Exponential Moving Average `public static void LinearNorm(DataChunkXY datain, bool zero, ref DataChunkXY dataout)` - Linear Normalization `public static void NonLinearNorm(DataChunkXY datain, bool zero, ref DataChunkXY dataout)` - Non-Linear Normalization

DataChynkXY.cs	Список точок, що обробляються
	Змінні
	`public int N;`
	`public List<Point2D> Chunk;`
	`public double xmin, xmax, ymin, ymax;`
	Конструктори
	`public DataChunkXY()` – конструктор за замовчуванням
	`public DataChunkXY(double [,] A)` – конструктор з параметрами
	Методи для обробки точок
	`public void Add(Point2D p)`
	`public void RemoveAt(int ind)`
	`public void ScaleX(double factor)`
	`public void ScaleY(double factor)`
	`public void FindExtremes()`
	Методи для роботи з файлами
	`static double GetDouble(string value, double def_value)`
	`static public DataChunkXY LoadTXT(string filename)`
	`public void SaveTXT(string filename)`

Розглянемо для ілюстрації роботу декількох методів.

Наприклад, метод Graph.DrawGrid класу Graph виводить зображення сітки графіку на форму

```
public void DrawGrid(Form form, Graphics g)
        {
            SolidBrush bcg_brush = new SolidBrush(bcg_color);
            Pen grid_pen = new Pen(grid_color,1);
            Pen axis_pen = new Pen(line_color, 2);

            g.FillRectangle(bcg_brush, top, left, width, height);

            int curx = 0, cury = 0;
            int count = 0;
            while (curx < width)
            {
                Point p1 = new Point(curx, top);
                Point p2 = new Point(curx, top + height);
                Point a1 = new Point(curx, y0 + 3);
                Point a2 = new Point(curx, y0 - 3);
                g.DrawLine(grid_pen, p1, p2);
                g.DrawLine(axis_pen, a1, a2);
                g.DrawString((count-5).ToString(), form.Font, Brushes.Black, new
PointF(curx-5,y0+5));
                count += 5;
                curx += grid_step;
            }
            while (cury < height)
            {
                Point p1 = new Point(left, cury);
                Point p2 = new Point(left + width, cury);
                g.DrawLine(grid_pen, p1, p2);
                cury += grid_step;
            }

            g.DrawLine(axis_pen, new Point(left, y0), new Point(left + width, y0));
            g.DrawLine(axis_pen, new Point(x0, top), new Point(x0, top+height));

            bcg_brush.Dispose();
            grid_pen.Dispose();
            axis_pen.Dispose();
               }
```

Метод Graph.DrawGrid визивається із головної форми методом Form1.PaintAll

```
private void PaintAll()
        {
            int w = pictureBox1.Width;
            int h = pictureBox1.Height;
            int upperN = w/step;

            if (initial_data != null && initial_data.N > 0)
            {
                if (initial_data.N * step > w)
                    w = (initial_data.N + 10) * step;
                upperN = initial_data.N;
            }

            //maximum possible width value
            if (w > 65534)
```

```
            {
                w = 65534;
                upperN = w / step;
            }

            if (this.WindowState != FormWindowState.Maximized)
                h = 400;

            bmp = new Bitmap(w, h);
            using (g = Graphics.FromImage(bmp))
            {
                graph = new Graph(0, 0, w, h);

                if (initial_data != null && initial_data.N > 0)
                {
                    graph.SetUnits(initial_data.xmin, initial_data.xmax, initial_data.ymin,
initial_data.ymax);
                    graph.DrawGrid(this, g);
                    graph.ArrangeY(this, g);

                    SolidBrush point_brush = new SolidBrush(Color.DarkOrange);
                    Pen lpen = new Pen(Color.Aqua, 1);
                    graph.DrawPoint(g, initial_data.Chunk[0], 5, point_brush);
                    for (int i = 1; i < upperN; i++)
                    {
                        graph.DrawPoint(g, initial_data.Chunk[i], 5, point_brush);
                        graph.DrawLine(g, initial_data.Chunk[i - 1], initial_data.Chunk[i],
lpen);
                    }

                    point_brush.Dispose();
                    lpen.Dispose();

                    textBox1.Text = initial_data.xmin.ToString();
                    textBox2.Text = initial_data.xmax.ToString();
                    textBox3.Text = initial_data.ymin.ToString();
                    textBox4.Text = initial_data.ymax.ToString();
                }
            }
            pictureBox1.Image = bmp;
        }
```

В результаті при завантаженні форми на ній прорисовується сітка (див.рис.2.4 вище).

Або статичний метод Preproc.SMA статичного класу Preproc забезпечує математичну обробку даних при лінійному згладжуванні

```
//-----------------------------------------------------------------------Simple Moving
Average
        public static void SMA(DataChunkXY datain, int period, ref DataChunkXY dataout)
        {
            if (datain != null && datain.N > period)
            {
                int N = datain.N;

                //-------left members
                for (int i = 0;i<period;i++)
                {
                    Point2D p = new Point2D(datain.Chunk[i].X, datain.Chunk[i].Y);
                    for (int j = i-1; j>-1;j--)
                    {
```

```
                    p.Y += datain.Chunk[j].Y;
                }
                p.Y /= 1.0 * (i + 1);
                dataout.Add(p);
            }

            //--------rest of members
            for (int i = period; i < N; i++)
            {
                Point2D p = new Point2D(datain.Chunk[i].X, datain.Chunk[i].Y);
                for (int j = i - 1; j > i-period-1; j--)
                {
                    p.Y += datain.Chunk[j].Y;
                }
                p.Y /= 1.0 * period;
                dataout.Add(p);
            }
            dataout.FindExtremes();
        }
    }
```

Метод Preproc.SMA визивається із головної форми методом Form1. simpleToolStripMenuItem_Click (вибір опції меню)

```
//------------------------------ SMA
    private void simpleToolStripMenuItem_Click(object sender, EventArgs e)
    {
        if (initial_data != null && initial_data.N > 0)
        {
            fmOneParamDialog setSMA = new fmOneParamDialog();
            setSMA.Owner = this;
            setSMA.Text = "Set period for SMA";
            setSMA.label1.Text = "SMA period";
            setSMA.ShowDialog();

            if (setSMA.ok)
            {
                int period = GetInt(setSMA.textBox1.Text, 3);
                DataChunkXY SMA_data = new DataChunkXY();
                Preproc.SMA(initial_data, period, ref SMA_data);

                fmGraphDialog showSMA = new fmGraphDialog(SMA_data);
                showSMA.Owner = this;
                showSMA.Text = "SMA Resuts";
                showSMA.Show();
            }
        }
        else
            MessageBox.Show("No Data Loaded", "Initial Data", MessageBoxButtons.OK,
MessageBoxIcon.Information);
    }
```

Результат обробки – список із згладженими даними SMA_Data типу DataChunkXY.

Таким чином, розроблені методи та форми надають користувачу зручні інструменти для візуалізації та обробки даних.

2.2. Організація навчання та первинного контролю за отриманими результатами

Для створення, навчання та первинного контролю за отриманими результатами призначено Модуль 2 пакету TSForecast.

Декомпозиція Етапу створення нейронної мережі – представлена на діаграмі, рис. 2.6.

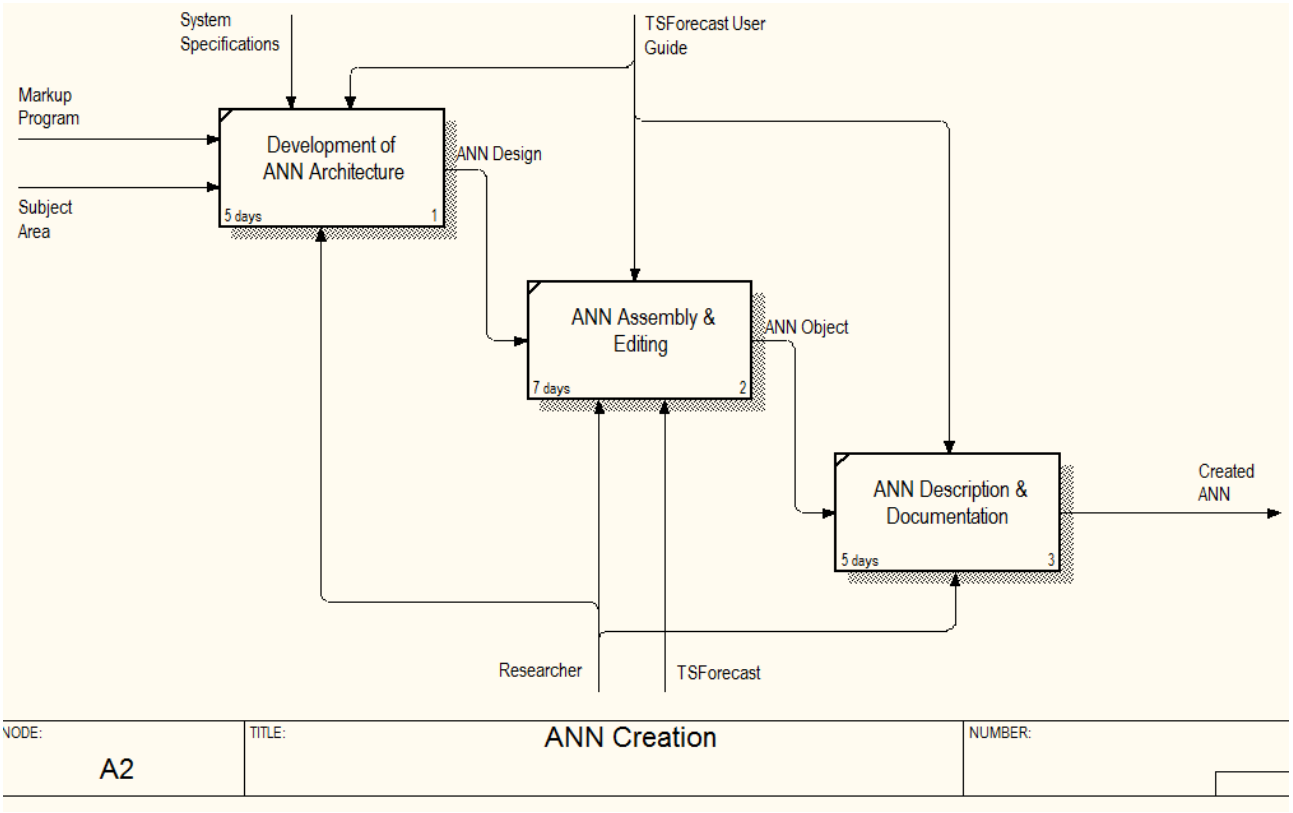


Рис.2.6. Декомпозиція етапу створення нейронної мережі

Підбір вдалої архітектури нейронної мережі, що дозволить класифікувати отримані сигнали, є нетривіальною задачею. В цілому, цей етап потребує значної кількості експериментальних випробувань із різними архітектурами та параметрами навчання. Тому важливим є надання зручних інструментів для швидкої зборки нейромережі з можливостями візуалізації її зв’язків та результатів навчання. Ці задачі виконує Модуль 2 пакету TSForecast.

Крім створення нейронної мережі, в Модулі 2 можна також провести її навчання. Декомпозиція етапу навчання показана на діаграмі рис.2.7.

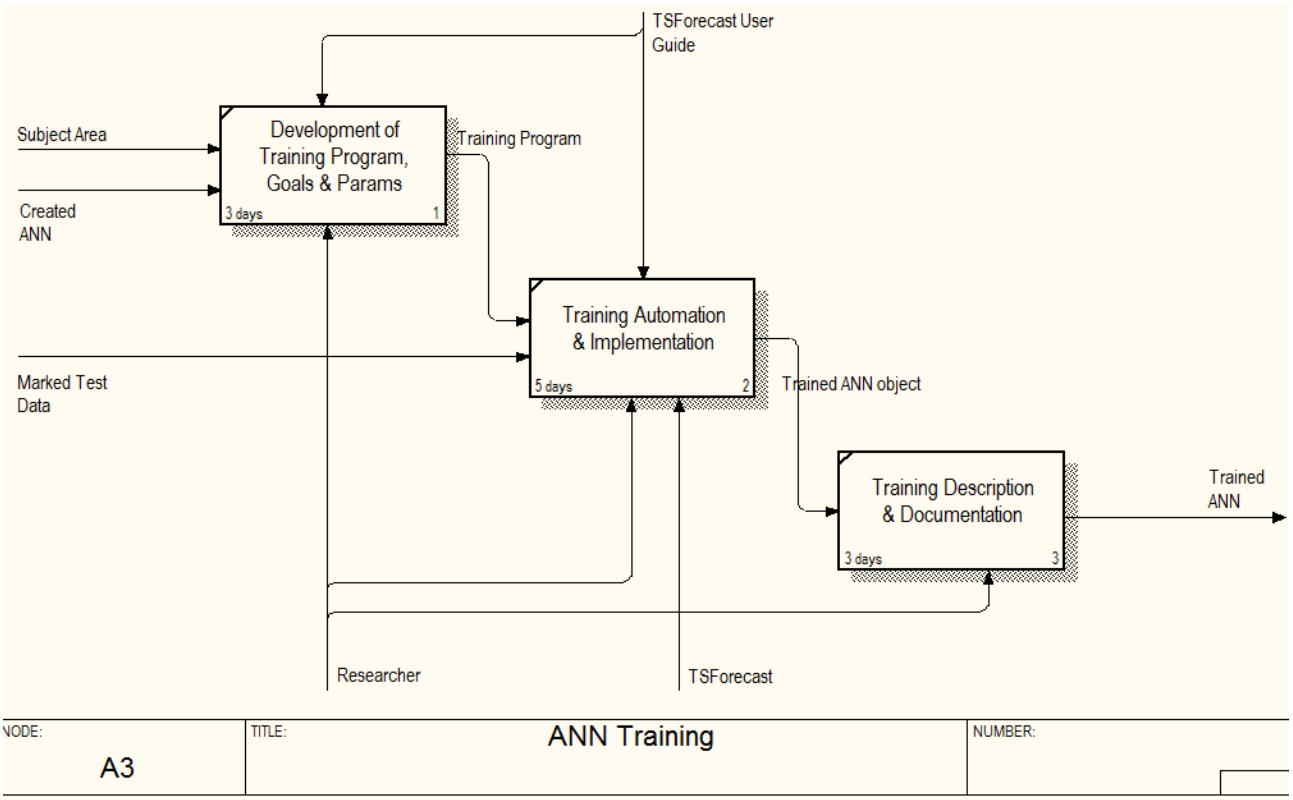


Рис.2.7. Декомпозиція підзадач етапу навчання нейронної мережі

Склад проекту TSF_Module2 представлений на рис.2.8.

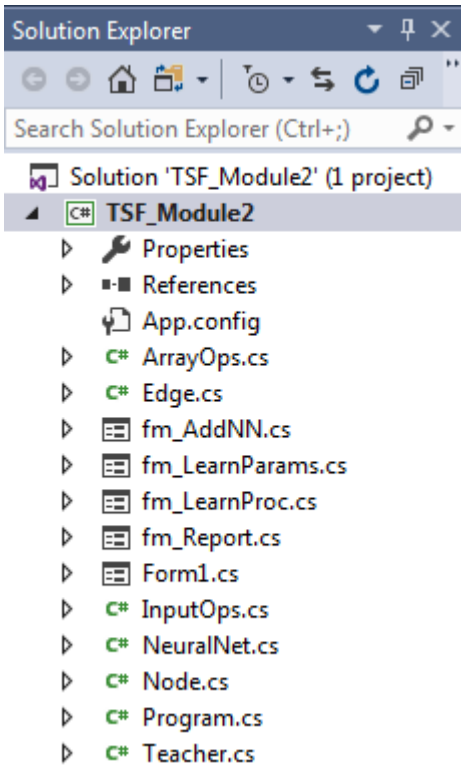


Рис.2.8. Склад проекту TSF_Module2

В проекті 5 класів-форм користувача та 6 класів-типів користувача. Їх перелік та короткий опис подано у табл.2.2. та нижче.

Таблиця 2.2

Класи-форми користувача, що наслідують Form.Windows

Клас	Опис, призначення

Form1.cs	Головна форма
	Змінні
	`NeuralNet ANN; //initial NN`
	`NeuralNet ANN2; //NN after learning`
	`//-------------painting`
	`Bitmap canvas;`
	`Graphics g;`
	`//-------------training`
	`int TE; //total number of epochs`
	`int BPE, SPB; //params for mini-batch: batches per epoch BPE, sample per batch SPB`
	`int tactics; //tactics: 1 - stochastic, 2 - batch, 3 - mini-batch`
	`string pos_folder, neg_folder; //folders with samples`
	`Teacher MarIvanna;`
	Призначення змінних описано в коментарях (вказівники на нейромережі та вчителя, візуалізація, параметри навчання)
	Методи
	`public Form1() – конструктор`
	`private void PaintANN(NeuralNet nn) – візуалізація нейромережі`
	Опції меню
	`private void addToolStripMenuItem_Click(object sender, EventArgs e)`
	`private void setParamsToolStripMenuItem_Click(object sender, EventArgs e)`
	`private void saveToolStripMenuItem_Click(object sender, EventArgs e)`
	`private void startToolStripMenuItem_Click(object sender, EventArgs e)`
	`private void reportToolStripMenuItem_Click(object sender, EventArgs e)`
	`private void openToolStripMenuItem_Click(object sender, EventArgs e)`
	і т.п.

Продовження таблиці 2.2

Клас	Опис, призначення

fmAddNN.cs	Форма для побудови нейромережі (кількість шарів, вузлів, функція активації)
	Змінні
	`int H;  //number of hidden layers`
	`public bool alldone;  //all OK`
	`public int[] nodes;  //numbers of nodes in layers`
	`public int func_num;  //number of activation function`
	Методи
	`public fm_AddNN() – конструктор`
	`private void ArrangeDG() – налаштування DataGrid для введення параметрів нейромережі`
	`private int SetFunc() – зчитування радіокнопок`
	`private void button1_Click(object sender, EventArgs e) – натиснення кнопки, задано кількість шарів, активація DataGrid`
	`private void button2_Click(object sender, EventArgs e) – створити нейромережу`
	`private void button3_Click(object sender, EventArgs e) – очистити все`

fmLearnParam.cs	Форма для визначення параметрів навчання (діалог)
	Змінні
	`public bool all_done;`
	`public int TE;  //total number of epochs`
	`public int BPE, SPB;  //params for mini-batch: batches per epoch BPE, sample per batch SPB`
	`public double eta, alpha;  //params for back propagation`
	`public int tactics;  //tactics: 1 - stochastic, 2 - batch, 3 - mini-batch`
	`public string pos_folder, neg_folder;  //folders with samples`
	Методи
	`public fm_LearnParams() – конструктор`
	`private void button1_Click(object sender, EventArgs e)` – знайти задану папку із позитивними зразками
	`private void button2_Click(object sender, EventArgs e) – знайти задану папку із негативними зразками`
	`private void radioButtons_CheckedChanged(object sender, EventArgs e) – обробка вибору радіо кнопок`
	`private void button3_Click(object sender, EventArgs e) – очистити все`
	`private void button4_Click(object sender, EventArgs e) – встановити визначені параметри`

Продовження таблиці 2.2

Клас	Опис, призначення

fmLearnProc.cs	Форма із статус-баром процесу навчання
	Змінні
	`public NeuralNet ANN;`
	`Teacher MarIvanna;`
	`Random rnd;`
	`public bool all_done;`
	Методи
	`public fm_LearnProc(NeuralNet _ANN, Teacher _MI)` – конструктор
	`private void button1_Click(object sender, EventArgs e)` – відміна навчання
	`private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)` – асинхронне виконання навчання
	`private void backgroundWorker1_ProgressChanged(object sender, ProgressChangedEventArgs e)` – подія зміна статусу
	`private void backgroundWorker1_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)` – завершення процесу, обробка виключень

fmReport.cs	Форма із звітами про навчання (графік та текст)
	Змінні
	`Graphics g;`
	`List<double> mse;`
	`List<string> report;`
	Методи
	`public fm_Report(NeuralNet ANN, Teacher MarIvanna)` – конструктор
	`private void DrawGraph(List<double> Y, int W, int H)` – візуалізація графіка навчання – залежність MSE від епохи
	`private void button1_Click(object sender, EventArgs e)` – зберегти графік
	`private void button2_Click(object sender, EventArgs e)` – зберегти текстовий звіт

Тепер розглянемо класи-типи користувача.

Основним типом є NeuralNet – нейромережа. Її змінні

```
public int NS { set; get; }         //num of nodes in the sensor layer (S-layer)
        public int H { set; get; }          //num of hidden layers
        public int []NH { set; get; }       //num of nodes in each hidden layer (A-layers)
        public int NR { set; get; }         //num of nodes in the reaction layer (R-layers)

        public int func { set; get; }       //index of activation function
                                            //1 - Sigmoid, 2 - Tanh, 3 - ELU

        public double eta { set; get; }     //learning rate
        public double alpha { set; get; }   //momentum

        public List<double[,]> W { set; get; }          //network (arrays - edges)
        public List<List<Point>> PP { set; get; }       //nodes on canvas to paint

        public List<double> MSE { set; get; }           //mean-square error per epoch
```

```
//---------------------actication function
delegate double F(double x);
delegate double FPrime(double x);

F f;              //activation function F(x)
FPrime fp;        //derivative F'(x)
//----------------------------------------
```

Діаграма класу, створена засобами VisualStudio, показана на рис.2.9.

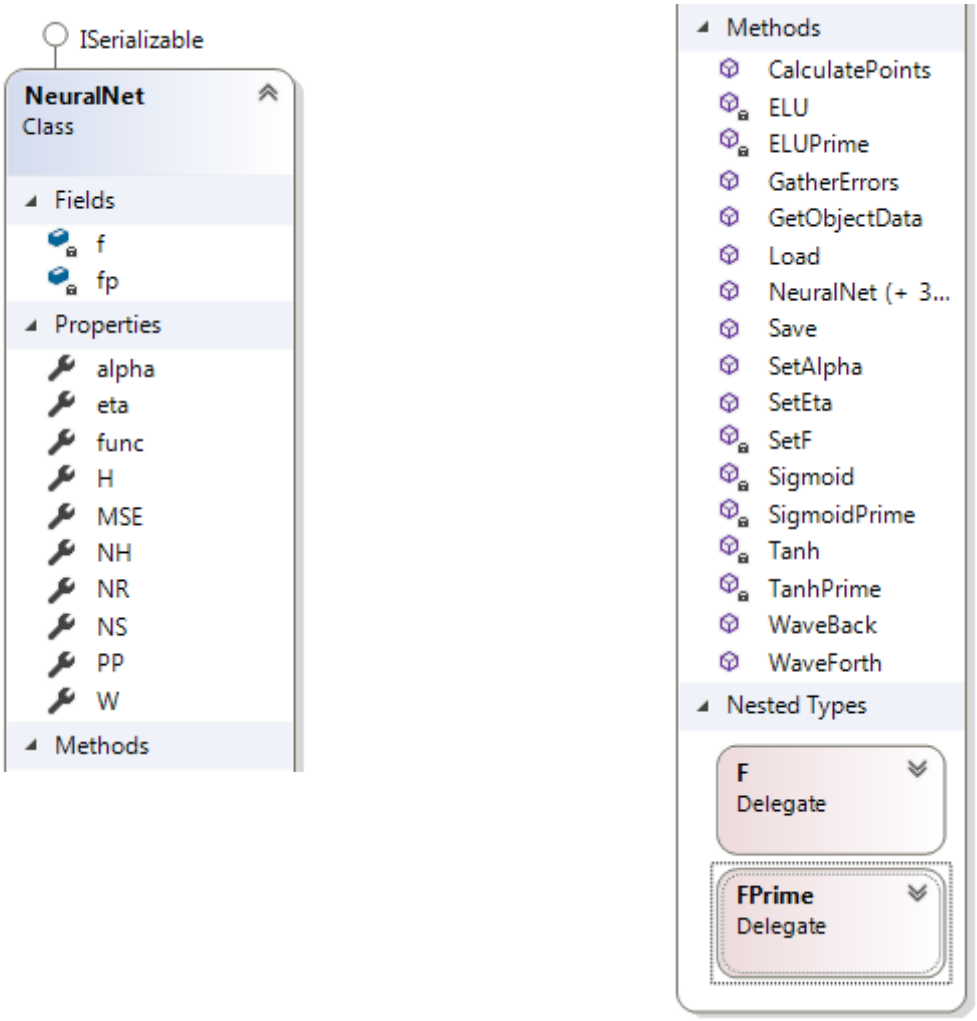


Рис.2.9. Діаграма основного класу проекту - NeuralNet

Методи класу NeuralNet

Три конструктори – за замовчуванням, з параметрами та для глибокої копії

```
public NeuralNet()
public NeuralNet(int _ns,int _h, int [] _nh, int _nr, int _f, double _eta, double _alpha)
public NeuralNet (NeuralNet nn)
```

Є також ще конструктори для відновлення об’єкта із файлу при десеріалізації

```
private NeuralNet(SerializationInfo info, StreamingContext ctx)
static public NeuralNet Load(string filename)
```

Клас NeuralNet реалізує інтерфейс ISerializable, що забезпечується

1) підключенням просторів імен

```
using System.Runtime.Serialization;
using System.Runtime.Serialization.Formatters.Binary;
```

2) нотацією

```
[Serializable]
    public class NeuralNet : ISerializable
```

3) методами

серіалізації (із об’єкту в бінарний потік)

```
public void GetObjectData(SerializationInfo info, StreamingContext ctx)
```

десеріалізації (із бінарного потоку в об’єкт)

```
private NeuralNet(SerializationInfo info, StreamingContext ctx)
```

Метод GetObjectData (серіалізація) має вигляд:

```
//--------------------------------------------------------------------в файл
        public void GetObjectData(SerializationInfo info, StreamingContext ctx)
        {
            info.AddValue("NS", this.NS);
            info.AddValue("H", this.H);
            for (int i = 0; i < H; i++)
                info.AddValue("NH_" + i.ToString(), this.NH[i]);
            info.AddValue("NR", this.NR);
            info.AddValue("func", this.func);
            info.AddValue("eta", this.eta);
            info.AddValue("alpha", this.alpha);

            int tot = W.Count;
            info.AddValue("Wcount", tot);
            for (int i = 0; i < tot; i++)
            {
                int nr = W[i].GetLength(0); //rows
                int nc = W[i].GetLength(1); //columns
                for (int j = 0; j < nr; j++)
                    for (int k = 0; k < nc; k++)
                        info.AddValue("W_" + i.ToString() + "_" + j.ToString() + "_" +
k.ToString(), this.W[i][j, k]);
            }

            int mse_count = MSE.Count;
            info.AddValue("MSEcount", mse_count);
            for (int i = 0; i < mse_count; i++)
                info.AddValue("MSE_" + i.ToString(), this.MSE[i]);
        }
```

Відповідно, конструктор для десеріалізації

```
//-----------------------------------------------------------из файла
        private NeuralNet(SerializationInfo info, StreamingContext ctx)
        {
            this.NS = info.GetInt32("NS");
            this.H = info.GetInt32("H");
            NH = new int[H];
            for (int i = 0; i < H; i++)
```

```
                NH[i] = info.GetInt32("NH_" + i.ToString());
            this.NR = info.GetInt32("NR");
            this.func = info.GetInt32("func");
            SetF(func);     //set activation function
            this.eta = info.GetDouble("eta");
            this.alpha = info.GetDouble("alpha");

            int tot = info.GetInt32("Wcount");
            W = new List<double[,]>();
            double[,] mesh = new double[NS + 1, NH[0]];
            W.Add(mesh);
            for (int i = 1; i < H; i++)
            {
                mesh = new double[NH[i - 1] + 1, NH[i]];
                W.Add(mesh);
            }
            mesh = new double[NH[H - 1] + 1, NR];
            W.Add(mesh);

            for (int i = 0; i < tot; i++)
            {
                int nr = W[i].GetLength(0); //rows
                int nc = W[i].GetLength(1); //columns
                for (int j = 0; j < nr; j++)
                    for (int k = 0; k < nc; k++)
                        this.W[i][j, k] = info.GetDouble("W_" + i.ToString() + "_" +
j.ToString() + "_" + k.ToString());
            }

            int mse_count = info.GetInt32("MSEcount");
            MSE = new List<double>();
            for (int i = 0; i < mse_count; i++)
                MSE.Add(info.GetDouble("MSE_" + i.ToString()));
        }
```

Сукупність чотирьох методів

```
public void GetObjectData(SerializationInfo info, StreamingContext ctx)
public void Save(string filename)
private NeuralNet(SerializationInfo info, StreamingContext ctx)
static public NeuralNet Load(string filename)
```

забезпечує зберігання об’єкту класу NeuralNet у бінарному файлі та його відновлення із файлу при завантаженні.

Низка методів класу NeuralNet призначена для встановлення та реалізації функцій активації та параметрів навчання

```
//-------------------reset learning params
        public void SetEta(double _eta)
        public void SetAlpha(double _alpha)
//-----------------------------------------------------activation function
       private void SetF(int f_num)
       private static double Sigmoid(double x)
       private static double SigmoidPrime(double x)
       private static double Tanh(double x)
```

```
private static double TanhPrime(double x)
private static double ELU(double x)
private static double ELUPrime(double x)
```

Пряме розповсюдження сигналу виконує метод

```
//move forth
   public void WaveForth(double []X, ref List<double[,]> S_in, ref List <double [,]> S_out)
```

Обернене розповсюдження

```
//move back
        public void WaveBack(double[] Y, List<double[,]> S_in, List<double[,]> S_out)
```

Збір відхилень з вузлів R-шару та підрахунок середнього квадратичного відхилення (MSE) виконує метод

```
//------------------------gather errors
        public double GatherErrors(double[] Y, List<double[,]> S_out)
        {
            double[,] sigma = new double[1, NR];
            int K = S_out.Count;

            double err = 0.0;
            //reaction layer
            for (int i = 0; i < NR; i++)
            {
                sigma[0, i] = Y[i] - S_out[K - 1][0, i];
                err += sigma[0, i] * sigma[0, i];
            }
            err /= NR;
            return err;
        }
```

Також в класі NeuralNet є метод

```
public void CalculatePoints(Point P0)
```

що призначений для розміщення нейромережі на екрані при візуалізації.

Навчання здійснюється об’єктом класу Teacher, котрий містить відповідні методи. Змінні класу Teacher

```
public int NE { set; get; }     //number of epoques

 public int NB { set; get; }     //number of batches per an epoque
 public int SB { set; get; }     //samples per a batch

 public int tactics { set; get; } //tactics

 public double eta { set; get; }
 public double alpha { set; get; }

 string pos_path;    //path to folder with positive samples
 string neg_path;    //path to folder with negative samples

 List<string> pos_samples;
List<string> neg_samples;
```

Діаграма класу представлена на рис.2.10.

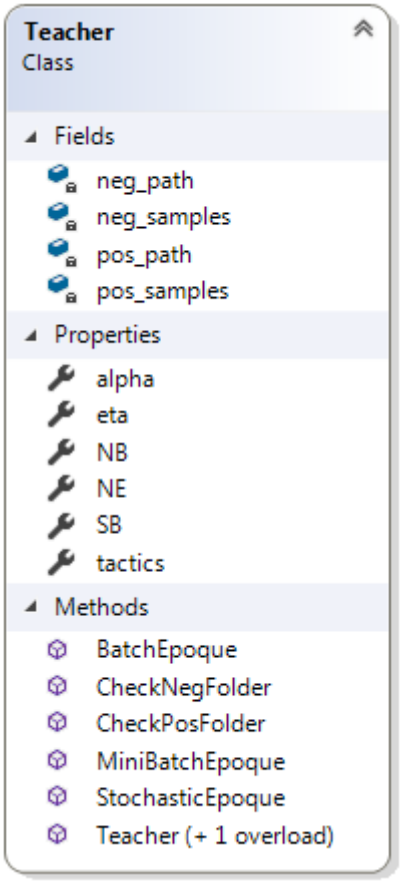


Рис.2.10. Клас Teacher для навчання нейромереж

Клас має два конструктори – з параметрами та для глибокої копії

```
//universal teacher
 public Teacher (int _tactics, int _NE, int _NB, int _SB, string _pos_path, string
_neg_path, double _eta, double _alpha)
//--------deep copy
 public Teacher(Teacher _MI)
```

При підключенні зразків до вчителя від перевіряє папки та підраховує кількість зразків

```
public void CheckPosFolder (int NS, ref int PQ, ref List<int> incompatible)
public void CheckNegFolder(int NS, ref int NQ, ref List<int> incompatible)
```

У класі вчителя також реалізуються відмінності в оберненому розповсюдженні помилки при навчанні за різними режимами.

Навчання може проводитися в одному із трьох режимів (Tactics of Gradient Descent):

– стохастичному (обернене розповсюдження в нейромережі відбувається після кожного зразка);
– упорядкованому (обернене розповсюдження в нейромережі відбувається по усередненим значенням в кінці епохи);
– змішаному (обернене розповсюдження в нейромережі відбувається по усередненим значенням після кожних m зразків, де m<N, менше загальної кількості, число вказується користувачем);

Відповідні методи класу Teacher

```
//--------------waveback for each
public void StochasticEpoque(NeuralNet ANN, Random rnd)
//----------------one waveback at the end
public void BatchEpoque(NeuralNet ANN, Random rnd)
//----------------periodic wavebacks in mini batches
public void MiniBatchEpoque(NeuralNet ANN, Random rnd)
```

Для забезпечення безпечного вводу та перетворення числових даних у проекті розроблено статичний клас InputOps. Його методи

```
//-------------------------string to double
public static double GetDouble(string value, double def_value, ref bool success)
//-------------------------string array to double array
public static double []GetDoubleArray(string []values, double def_value, ref bool success)
//-------------------------string to int
public static int GetInt(string value, int def_value, ref bool success)
```

Також, цей клас містить два методи для перемішування рядків у списках файлів

```
//---------------------------shuffle list - extension method
public static void Shuffle<T>(this IList<T> list, Random rnd)
//-----------------------------swap int list - extension method
public static void Swap<T>(this IList<T> list, int i, int j)
```

ці методи розроблені як розширення для інтерфейсу IList.

Приклад використання – у методах для оберненого розповсюдження помилки у класі Teacher. Наприклад, при стохастичному режимі

```
//--------------waveback for each
        public void StochasticEpoque(NeuralNet ANN, Random rnd)
        {
            int PS = pos_samples.Count;
            int NS = neg_samples.Count;

            //shuffling positions
            List<int> numpos = Enumerable.Range(0, PS).ToList();
            List<int> numneg = Enumerable.Range(0, NS).ToList();
            numpos.Shuffle(rnd);
            numneg.Shuffle(rnd);
```

```
        int NMAX = Math.Max(PS, NS);
        ANN.SetEta(eta);
        ANN.SetAlpha(alpha);

        //two nodes in R-layer
        double[] YPos = new double[] { 0.9, 0.1 };
        double[] YNeg = new double[] { 0.1, 0.9 };

        //enlarge outputs for tanh
        if(ANN.func == 2)
        {
            YPos[1] = -0.9;
            YNeg[0] = -0.9;
        }

        List<double> errors = new List<double>();

        for (int i = 0;i<NMAX;i++)
        {
            //positive sample
            List<double[,]> S_in = new List<double[,]>();
            List<double[,]> S_out = new List<double[,]>();

            string[] XS = File.ReadAllLines(pos_samples[numpos[i % PS]]);
            bool success = true;
            double[] X_in = InputOps.GetDoubleArray(XS, 0, ref success);
            ANN.WaveForth(X_in, ref S_in, ref S_out);
            double err = ANN.GatherErrors(YPos, S_out);
            errors.Add(err);
            ANN.WaveBack(YPos, S_in, S_out);

            //negative sample
            S_in = new List<double[,]>();
            S_out = new List<double[,]>();

            string []XSneg = File.ReadAllLines(neg_samples[numneg[i % NS]]);
            success = true;
            X_in = InputOps.GetDoubleArray(XSneg, 0, ref success);
            ANN.WaveForth(X_in, ref S_in, ref S_out);
            err = ANN.GatherErrors(YNeg, S_out);
            errors.Add(err);
            ANN.WaveBack(YNeg, S_in, S_out);
        }

        //------find mse per epoch
        double mse = 0.0;
        int K = errors.Count;
        for (int i = 0; i < K; i++)
            mse += errors[i];
        mse /= K;

        ANN.MSE.Add(mse);
    }
```

В цілому, сукупність представлених рішень у проекті TSF_Module2 надає користувачу зручний інтерфейс для проведення експериментів із навчання нейронних мереж різноманітних архітектур з різними параметрами та режимами.

2.3. Процедура верифікації нейронних мереж

Після навчання, нейронна мережа має пройти процедуру верифікації на зразках, що не були їй пред'явлені при навчанні. Показники верифікації розраховуються за формулами, представленими у табл.1.3.

Декомпозиція підзадач етапу верифікації представлена на діаграмі, рис. 2.11.

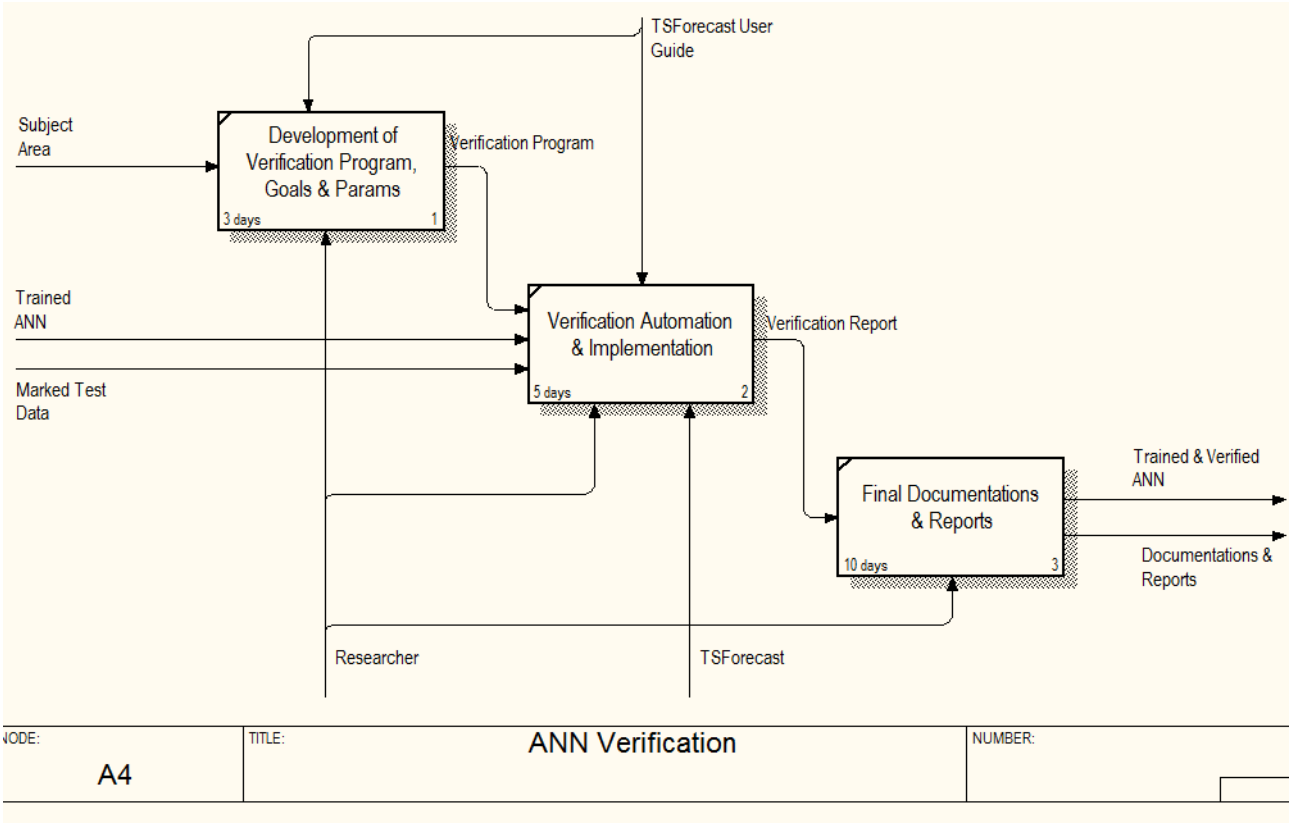


Рис.2.11. Декомпозиція підзадач етапу верифікації навченої нейромережі

Для проведення верифікації у пакеті TSForecast призначено Модуль 3. Структура проекту TSF_Module 3 показана на рис. 2.12.

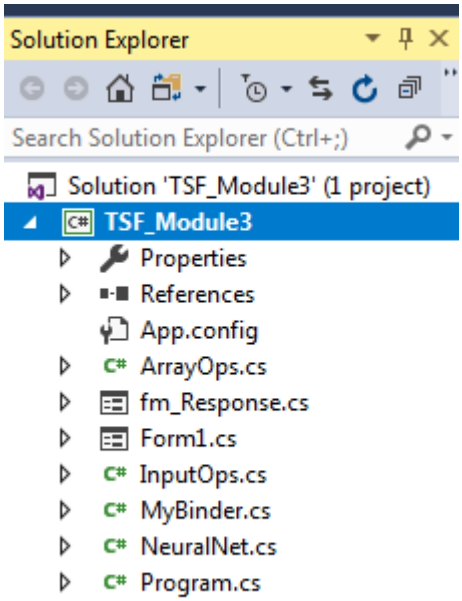


Рис.2.12. Структура проекту TSF_Module 3

В проекті два класи-форми користувача та 4 класи-типи. Опис змінних та методів цих класів подано у табл.2.3.

Таблиця 2.3

Класи користувача в проекті TSF_Module 3

Клас	Опис, призначення
Класи-форми, що наслідують Form.Windows	

Form1.cs	Головна форма
	Змінні
	`NeuralNet ANN;`
	`string pos_folder, neg_folder;  //folders with samples`
	`List<string> pos_response;  //info on NN responses`
	`List<string> neg_response;`
	`List<string> report; //report to save`
	Методи
	`public Form1() – конструктор`
	`private void button2_Click(object sender, EventArgs e) – завантажити папку с позитивними зразками`
	`private void button3_Click(object sender, EventArgs e) – завантажити папку з негативними зразками`
	`private void button4_Click(object sender, EventArgs e) – очистити форму`
	`private void button5_Click(object sender, EventArgs e) – процедура верифікації`
	`private void button6_Click(object sender, EventArgs e) – показати сигнали на виході`
	`private void button7_Click(object sender, EventArgs e) – зберегти звіт`

Продовження таблиці 2.3

Клас	Опис, призначення
fmResponse.cs	Форма з інформацією про сигнали на виході нейромережі Методи `public fm_Response(List<string> pos_response, List<string> neg_response)- конструктор`
	Класи-типи користувача
NeuralNet.cs	Нейронна мережв Клас детально описаний в Розділі 2.2

ArrayOps.cs	Статичний клас зі статичними методами для обробки масивів даних
	Методи
	`//res = coef * vector`
	`public static void Coef_Vector_Multi(double coef, double[] vector, ref double[] res)`
	`//res = coef * array`
	`public static void Coef_Arr_Multi(double coef, double[,] arr, ref double[,] res)`
	`//res = array * array`
	`public static void Arr_Arr_Multi(double[,] arr1, double[,] arr2, ref double[,] res)`
	`//res = array T`
	`public static void Arr_Transpose(double[,] arr, ref double[,] res)`

InputOps.cs	Статичний клас зі статичними методами для обробки вводу числових даних
	Методи
	`//-------------------------string to double`
	`public static double GetDouble(string value, double def_value, ref bool success)`
	`//-------------------------string array to double array`
	`public static double[] GetDoubleArray(string[] values, double def_value, ref bool success)`
	`//-------------------------string to int`
	`public static int GetInt(string value, int def_value, ref bool success)`
	`//-----------------shuffle list - extension method`
	`public static void Shuffle<T>(this IList<T> list, Random rnd)`
	`//-------------swap int list - extension method`
	`public static void Swap<T>(this IList<T> list, int i, int j)`

MyBinder.cs	Клас що пере визначає метод прив’язки типу до зборки Метод `public override Type BindToType(string assemblyName, string typeName)`

Для того, щоб об’єкти класу NeuralNet, створені у Модулі 2, змогли відновлюватися із бінарних файлів у Модулі 3, створено клас MyBinder, котрий перевизначає метод BindToType для SerializationBinder

```
sealed class MyBinder : SerializationBinder
{
    public override Type BindToType(string assemblyName, string typeName)
    {
        Type typeToDeserialize = null;

        /*
        // For each assemblyName/typeName that you want to deserialize to
        // a different type, set typeToDeserialize to the desired type.
        String exeAssembly = Assembly.GetExecutingAssembly().FullName;


        // The following line of code returns the type.
        typeToDeserialize = Type.GetType(String.Format("{0}, {1}",
            typeName, exeAssembly));
        */
        assemblyName = Assembly.GetExecutingAssembly().FullName;


        if (assemblyName ==
        "TSF_Module2, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null")
    {
            typeToDeserialize = Type.GetType(String.Format("{0}, {1}",
                typeName, assemblyName));
            return typeToDeserialize;
        }

        if (typeName ==  "TSF_Module2.NeuralNet")
    {
            typeName = typeName.Replace("TSF_Module2", "TSF_Module3");
            typeToDeserialize = Type.GetType(String.Format("{0}, {1}", typeName,
assemblyName));
            return typeToDeserialize;
        }

        return typeToDeserialize;
    }
}
```

При десеріалізації в бінарному файлі з об’єктом класу NeuralNet заміщується назва зборки

```
"TSF_Module2, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
```

оскільки він створений у Модулі 2, на поточну назву (Модуля 3).

Також, в заголовках бінарного файлу заміщується ім’я проекту, в якому десеріалізується об’єкт

```
typeName = typeName.Replace("TSF_Module2", "TSF_Module3");
```

В результаті, біндер дозволяє завантажувати і працювати з файлами *.ann, котрі містять об’єкти типу NeuralNet.

Власне, процедура верифікації - це метод головної форми Form1

```
private void button5_Click(object sender, EventArgs e)
```

що починається із завантаження списків файлі із обраних папок

```
string[] files = Directory.GetFiles(pos_folder);
List<string> pos_samples = files.ToList();

files = Directory.GetFiles(neg_folder);
List<string> neg_samples = files.ToList();

int PS = pos_samples.Count;
int NS = neg_samples.Count;
```

Далі створюються списки векторів сигналів на виході нейромережі при реакції на позитивні та негативні зразки

```
pos_response = new List<string>();
neg_response = new List<string>();
```

Зразки по черзі визиваються за іменем файлу, подаються на вхід нейронної мережі для розпізнавання, з двох вузлів на виході знімаються вихідні сигнали

```
//positive samples
for(int i = 0;i<PS;i++)
{
    List<double[,]> S_in = new List<double[,]>();
    List<double[,]> S_out = new List<double[,]>();

    string[] XS = File.ReadAllLines(pos_samples[i]);
    bool success = true;
    double[] X_in = InputOps.GetDoubleArray(XS, 0, ref success);
    ANN.WaveForth(X_in, ref S_in, ref S_out);

    //two nodes in R-layer
    int K = S_out.Count;
    if (S_out[K - 1][0, 0] > S_out[K - 1][0, 1])
        TP++;
    else
        FN++;

    pos_response.Add(S_out[K - 1][0, 0].ToString());
    pos_response.Add(S_out[K - 1][0, 1].ToString());
    pos_response.Add(String.Empty);
}
```

Аналогічно – для негативних сигналів.

Далі відбувається розрахунок показників за зібраною статистикою

```
double TPR = 1.0 * TP / PS;
double TNR = 1.0 * TN / NS;
double ACC = 1.0 * (TP + TN) / (PS + NS);
```

```
            double MCC = 1.0 * (TP * TN - FP * FN) / Math.Sqrt(PS * NS * (TP + FP) * (TN +
FN));
```

та розміщення результатів по контролам форми

```
//------output to boxes
            textBox4.Text = TP.ToString();
            textBox5.Text = TN.ToString();
            textBox6.Text = FP.ToString();
            textBox7.Text = FN.ToString();

            textBox8.Text = TPR.ToString("F5");
            textBox9.Text = TNR.ToString("F5");
            textBox10.Text = ACC.ToString("F5");
            textBox11.Text = MCC.ToString("F5");
```

В кінці методу формується рядок звіту за отриманими результатами

```
//-----------report
            report = new List<string>();
            report.Add("TP = " + TP.ToString());
            report.Add("TN = " + TN.ToString());
            report.Add("FP = " + FP.ToString());
            report.Add("FN = " + FN.ToString());
            report.Add("TPR = " + TPR.ToString());
            report.Add("TNR = " + TNR.ToString());
            report.Add("ACC = " + ACC.ToString());
            report.Add("MCC = " + MCC.ToString());
            if (pos_response.Count > 1 && neg_response.Count > 1)
            {
                report.Add("Examples of response:");
                report.Add("On a positive sample:" + pos_response[0] + "; " +
pos_response[1]);
                report.Add("On a negative sample:" + neg_response[0] + "; " +
neg_response[1]);
            }
```

Верифікація дозволяє стандартизувати процедуру вибору кращих із навчених нейромереж, а також відбракувати не досить вдалі спроби.

Таким чином, верифікація є необхідним етапом для впровадження технологій штучного інтелекту у виробництво.

2.4. Реалізація метаструктури - сумісна робота ансамблю навчених нейронних мереж у каскаді

Заключним етапом у проекті, що має на меті розпізнавання декількох сигналів з високою точністю, є побудова метаструтури із навчених на парах сигналів нейромереж.

Для цього призначено Модуль 4 пакету TSForecast.

Структура проекту представлена на рис.2.13.

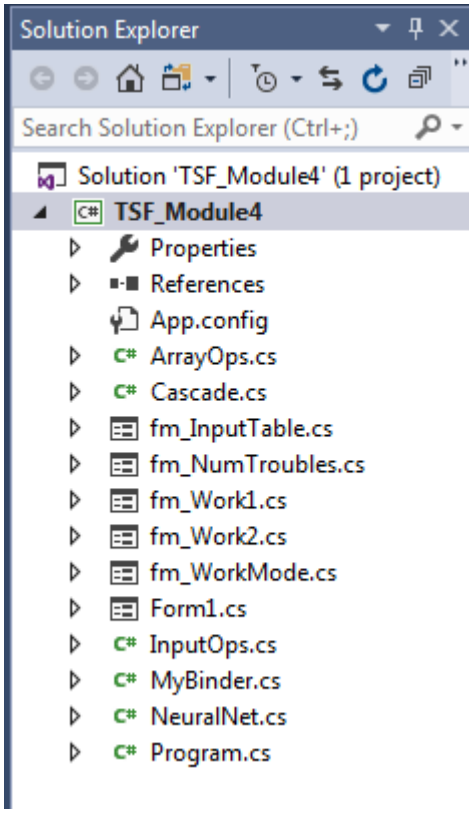


Рис.2.13. Структура проекту TSF_Module4

Ми зустрічаємо вже знайомі нам класи-типи користувача: NeuralNet – нейромережа, MyBinder – біндер для зв’язку класів у різних проектах для десеріалізації об’єктів із бінарних файлів, ArryOps – статичні методи обробки масивів, InputOps – статичні методи обробки числових даних на вході.

Основним типом даних у Модулі 4 є Cascade – каскад. Це - метаструктура, побудована із декількох нейромереж навчених на парах сигналів. Завдяки процедурі «голосування» при оцінюванні отриманого

сигналу, каскад здатен розпізнавати всю множину сигналів, яку здатні розпізнати його вузли – нейромережі.

Клас Cascade має наступні властивості

```
public List<List<string>> Networks { set; get; }
public List<string> Samples { set; get; }
public int K { set; get; }
```

Таким чином, у об’єкті цього класу зберігаються не самі нейромережі та зразки, а тільки повні шляхи до відповідних файлів.

Після фізичного зчитування даних із файлів (процедура під’єднання - Connect), вони зберігаються у приватних змінних класу

```
List<List<NeuralNet>> ANNs;
List<List<double[]>> signals;
```

Діаграма класу, створена засобами Visual Studio, представлена на рис. 2.14.

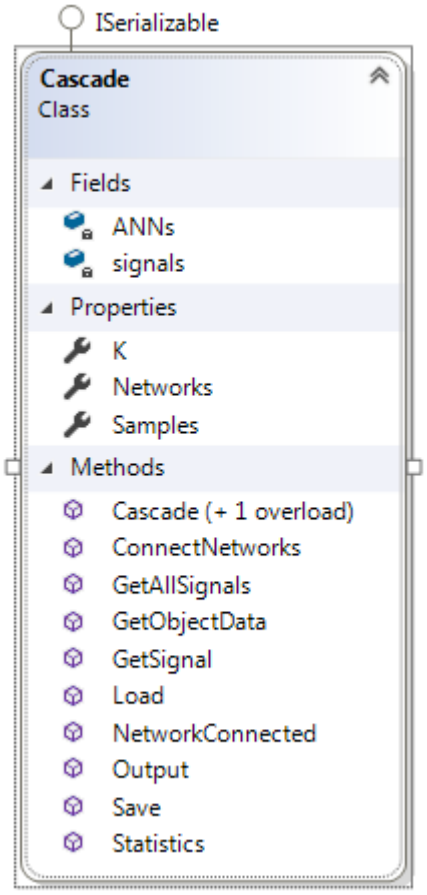


Рис.2.14. Діаграма класу Cascade

Конструктор класу

```
public Cascade(int _K, List<List<string>> _networks, List<string> _samples)
        {
            K = _K;

            Networks = new List<List<string>>();
            Samples = new List<string>();

            for (int i = 0; i < K + 1; i++)
            {
                Networks.Add(new List<string>());
                for (int j = 0; j < K + 1; j++)
                    Networks[i].Add(_networks[i][j]);

                Samples.Add(_samples[i]);
            }
        }
```

Оскільки об’єкти класу Cascade мають зберігатися у файлах та завантажуватися для повторного застосування, клас реалізує інтерфейс ISerializable

```
[Serializable]
public class Cascade : ISerializable
```

При серіалізації об’єкт перетворюються у бінарний потік у методі

```
//--------------------------------------------------------------------в файл

public void GetObjectData(SerializationInfo info, StreamingContext ctx)
        {
            info.AddValue("K", this.K);
            for (int i = 0; i < K + 1; i++)
            {
                for (int j = 0; j < K + 1; j++)
                    info.AddValue("Networks_" + i.ToString() + "_" + j.ToString(),
this.Networks[i][j]);
            }

            for (int i = 0; i < K + 1; i++)
                info.AddValue("Samples_" + i.ToString(), this.Samples[i]);
        }
```

Відповідно, при десеріалзації об’єкт відновлюється із бінарного потоку у методі (конструкторі)

```
    //-----------------------------------------------------------из файла
        private Cascade(SerializationInfo info, StreamingContext ctx)
        {
            this.K = info.GetInt32("K");

            Networks = new List<List<string>>();
            Samples = new List<string>();

            for (int i = 0; i < K + 1; i++)
            {
                Networks.Add(new List<string>());
                for (int j = 0; j < K + 1; j++)
                    Networks[i].Add(info.GetString("Networks_" + i.ToString() + "_" +
j.ToString()));
```

```
                Samples.Add(info.GetString("Samples_" + i.ToString()));
            }
        }
```

«Голосування» в каскаді відбуваються за більшістю відданих за даний сигнал «голосів» нейромереж, що входять у каскад. Це реалізується у методі Cascade.Output

```
//-----------------------work
        public int Output (double []signal)
        {
            int result = -1;

            double[] piggybank = new double[K + 1];
            for (int i = 0; i < K + 1; i++)
                piggybank[i] = 0;

            for (int i = 0; i< K; i++) //meaning rows
            {
                for (int j = i+1;j<K+1;j++) //meaning columns
                {
                    List<double[,]> S_in = new List<double[,]>();
                    List<double[,]> S_out = new List<double[,]>();

                    ANNs[i][j].WaveForth(signal, ref S_in, ref S_out);
                    // two nodes in R - layer
                    int M = S_out.Count;

                    //which is bigger
                    if (S_out[M - 1][0, 0] > S_out[M - 1][0, 1])
                        piggybank[i] += S_out[M - 1][0, 0]; //i - positive
                    else
                        piggybank[j] += S_out[M - 1][0, 1]; //j - negative
                }
            }
            double max = piggybank.Max(); //piggy voting
            result = Array.IndexOf(piggybank, max);
            return result;
        }
```

Метод збору статистики нагадує процедуру верифікації, де кожен скалярний параметр тепер перетворюється у вектор

```
//-----------------------------------catch statistics
        public void Statistics(ref int[] T, ref int[] F, ref int[] Count)
        {
            int N = signals.Count;   //N = K+1
            for(int i = 0; i<N; i++)
            {
                int M = signals[i].Count;
                for(int j = 0; j<M;j++)
                {
                    int echo = Output(signals[i][j]);
                    if (echo == i)
                        T[i]++;
                    else
                        F[echo]++;
                }
                Count[i] = M;
            }
           }
```

Зберігаються каскади у файлах із розширенням *ccd (рис.2.15).

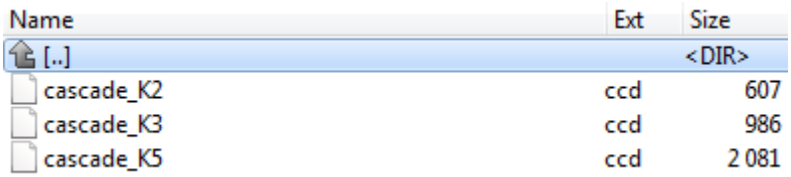


Рис.2.15. Папка із декількома зібраними каскадами (індекс вказує на кількість неполадок, що здатен розрізняти каскад, окрім штатної роботи)

Цікавим проектним рішенням є організація завантаження нейромереж для формування каскаду. Щоб користувач не заплутався та міг підключити кожну нейромережу до відповідного вузла каскаду, на формі формується таблиця, аналогічна табл.1.6 (див. рис.3.51). По мірі завантаження нейромереж, знаки питання змінюються на «галочки».

Таблиця із знаками питання та відмітками про вибір є зображенням, що формується у PictureBox. Обробка зображення здійснюється у методі fm_InputTable. ArrangeForm таким чином

```
//-------------picture box
            pictureBox1.Size = new Size(pb_size, pb_size);
            bmp = new Bitmap(pb_size, pb_size);
            Pen dark_gray_pen = new Pen(Color.DarkGray, 2);
            SolidBrush dark_orange = new SolidBrush(Color.DarkOrange);

            string[] headers = new string[K + 1];
            headers[0] = "P";
            for (int i = 1; i < K + 1; i++)
                headers[i] = "N" + i.ToString();

            using (g = Graphics.FromImage(bmp))
            {
                //drawing grid
                for (int i = 0; i < K + 2; i++)
                {
                    Point p_up = new Point(i * a, 0);
                    Point p_down = new Point(i * a, pb_size);
                    g.DrawLine(dark_gray_pen, p_up, p_down);
                }
                for (int i = 0; i < K + 2; i++)
                {
                    Point p_left = new Point(0, i * a);
                    Point p_right = new Point(pb_size, i * a);
                    g.DrawLine(dark_gray_pen, p_left, p_right);
                }

                //fill unused squares
                for (int i = 1; i < K + 2; i++)
                    for(int j = 0;j<i;j++)
                    {
```

```
                    Point p_top_left = new Point((j + 1) * a, i * a);
                    Rectangle rec = new Rectangle(p_top_left, new Size(a,a));
                    g.FillRectangle(dark_orange, rec);
                }

            //write headers
            for (int i = 0; i < K + 1; i++)
            {
                Point p_text = new Point((i + 1) * a + a / 4, a / 4);

                g.DrawString(headers[i], text_font, Brushes.Brown, p_text);
            }

            for (int i = 0; i < K + 1; i++)
            {
                Point p_text = new Point(a / 4, (i + 1) * a + a / 4);
                g.DrawString(headers[i], text_font, Brushes.Brown, p_text);
            }

            //write questions
            for (int i = 1; i < K + 1; i++)
            {
                for (int j = i+1; j < K+2; j++)
                {
                    Point p_text = new Point(j * a + a / 4, i * a + a / 4);
                    g.DrawString("?", text_font, Brushes.OrangeRed, p_text);
                }
            }
        }
        pictureBox1.Image = bmp;
```

Подвійний клік на комірці таблиці обробляється у методі

```
//----------------------------------------load ANN file
        private void pictureBox1_MouseDoubleClick(object sender, MouseEventArgs e)
        {
            //0 - positive; 1,2,3... - index of negative
            int row = e.Y / a - 1;
            int column = e.X / a - 1;

            string anns = String.Empty;
            if (row == 0)
                anns = "P - ";
            else
                anns = "N" + row.ToString() + " - ";
            anns += "N" + column.ToString();

            OpenFileDialog load_nn = new OpenFileDialog();
            load_nn.Filter = "Artificial Neural Network|*.ann";
            load_nn.Title = "Load Network: " + anns;

            if (load_nn.ShowDialog() == DialogResult.OK)
            {
                //write to list
                networks[row][column] = load_nn.FileName;

                //change "?" to "V"
                Pen green_pen = new Pen(Color.MediumSpringGreen, 2);
                using (g = Graphics.FromImage(bmp))
                {
                    //erase "?"
                    Point p0 = new Point((column + 1) * a, (row + 1) * a);
                    Point p_top_left = new Point(p0.X + 10, p0.Y + 10);
```

```
                    Rectangle rec = new Rectangle(p_top_left, new Size(3 * a / 4, 3 * a /
4));
                    g.FillRectangle(Brushes.White, rec);

                    //draw "V"
                    Point p1 = new Point(p0.X + 3 * a / 8, p0.Y + a / 2);
                    Point p2 = new Point(p0.X + a / 2, p0.Y + 5 * a / 8);
                    Point p3 = new Point(p0.X + 3 * a / 4, p0.Y + a / 4);
                    g.DrawLine(green_pen, p1, p2);
                    g.DrawLine(green_pen, p2, p3);
                }
                pictureBox1.Image = bmp;
            }
        }
```

Малюнок змінюється і шлях до файлу з нейромережею підключається до вузлу каскаду. В результаті, навіть при десятках нейронних мереж, що можуть бути підключені, користувач зможе впевнено знайти відповідний вузол.

Демонстрація роботи каскаду з окремим сигналом здійснюється у формі fmWork1. Сигнал обирається із завантаженого пулу довільно. Вид сигналу відомий, оскільки його закодовано у назві папки. Але каскаду це невідомо, і він самостійно розпізнає сигнал. При якісно навчених нейромережах-вузлах розпізнавання проходить успішно в більшості випадків.

За результатами розпізнавання у формі fmWork1 формується зображення (див.рис. 3.59 та 3.62), яке можливо зберегти у трьох форматах (jpg, bmp, png). Зберігання виконує метод

```
//--------------------------save picture
        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog save_image = new SaveFileDialog();
            save_image.Filter = "Bitmaps|*.bmp|PNG files|*.png|JPEG files|*.jpg";
            save_image.Title = "Save image of network/networks";
            ImageFormat format = ImageFormat.Jpeg;

            if (save_image.ShowDialog() == DialogResult.OK)
            {
                switch (save_image.Filter)
                {
                    case "*.bmp":
                        format = ImageFormat.Bmp;
                        break;
                    case "*.png":
                        format = ImageFormat.Png;
                        break;
                }
                //-----------white background
                Bitmap white = new Bitmap(canvas.Width, canvas.Height);
                using (g = Graphics.FromImage(white))
                {
                    g.Clear(Color.White);
                    g.DrawImage(canvas, 0, 0);
```

```
            }
            white.Save(save_image.FileName, format);
        }
```

Розпізнавання багатьох сигналів із папок, що підключаються до каскаду, проходить із збором статистики та підрахунком показників, аналогічним до процедури верифікації однієї нейромережі.

За отриманими результатами формується звіт

```
//-----------report
            report = new List<string>();
            report.Add("TP = " + TP.ToString());
            report.Add("TN = " + TN.ToString());
            report.Add("FP = " + FP.ToString());
            report.Add("FN = " + FN.ToString());
            report.Add("TPR = " + TPR.ToString());
            report.Add("TNR = " + TNR.ToString());
            report.Add("ACC = " + ACC.ToString());
            report.Add("MCC = " + MCC.ToString());
            report.Add(String.Empty);
            report.Add("K = " + K.ToString());
            for (int i = 1; i < K + 1; i++)
            {
                report.Add("N" + i.ToString());
                report.Add("     True = " + T[i].ToString() + ";  False = " +
F[i].ToString() + "; Count = " +
                    C[i].ToString());
                report.Add(String.Empty);
            }
```

Звіт можна зберегти у текстовому файлі

```
//---------------------------------save statistics
        private void saveToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog save_report = new SaveFileDialog();
            save_report.Filter = "Text file|*.txt";
            save_report.Title = "Save work test results";
            if (save_report.ShowDialog() == DialogResult.OK)
            {
                using (StreamWriter sw = new StreamWriter(save_report.FileName, false,
Encoding.UTF8))
                {
                    foreach (string s in report)
                    {
                        sw.WriteLine(s);
                    }
                }
            }
        }
```

Таким чином, користувач отримує оригінальний інструмент для побудови каскадів із нейромереж, котрий не має аналогів у світі.

Проведені чисельні експерименти показали, що каскади дозволяють розпізнавати множинні сигнали із високою точністю. Тому побудова таких систем є перспективним напрямом досліджень в галузі штучного інтелекту.

РОЗДІЛ 3. АНАЛІЗ ОТРИМАНИХ РЕЗУЛЬТАТІВ

3.1. Результати обробки первинних даних та отримання згортки для навчання нейронних мереж

Необхідні скрипти, розроблені в MATLAB, що забезпечує швидку обробку матричних даних, подані у Додатку Б.

Для нарізки довгого файлу з даними на фрагменти заданої довжини із заданим кроком запускаємо скрипт frags.m і обираємо файл з даними у вікні файлового діалогу (рис.3.1).

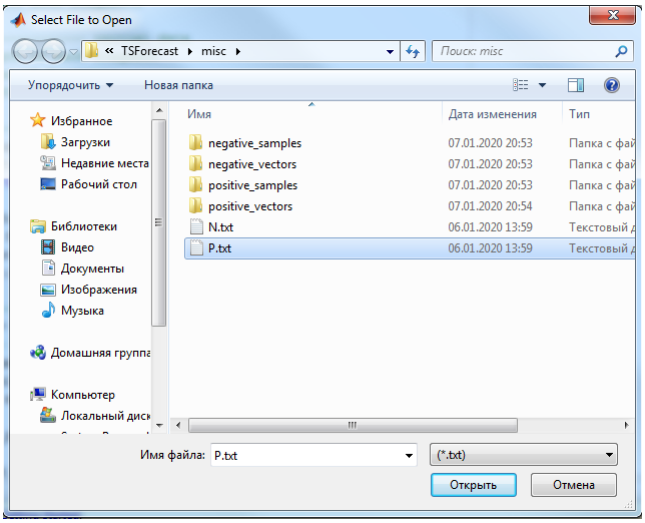


Рис.3.1. Вибір файлу з даними для нарізки

Далі, обираємо папку для запису фрагментів (рис.3.2).

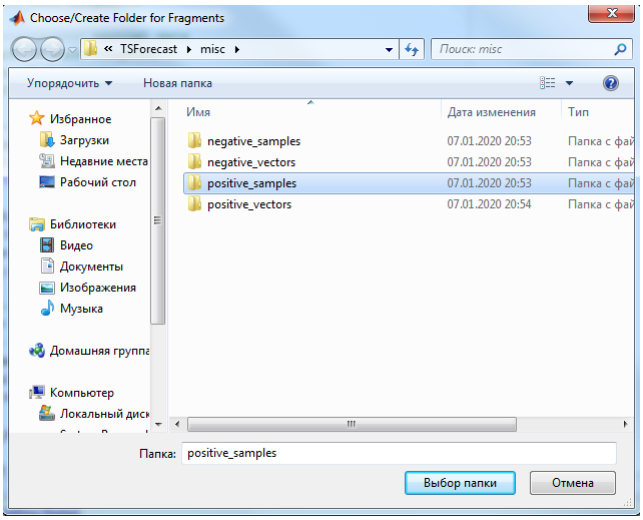


Рис.3.2. Призначення папки для збереження отриманих фрагментів

Через декілька секунд отримуємо повідомлення про успішне виконання (рис.3.3).

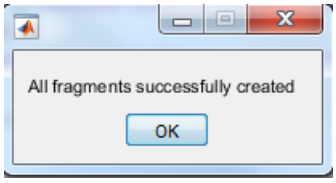


Рис.3.3. Повідомлення про успішне виконання нарізки

Всі файли отримують ім’я та нумеруються автоматично і зберігаються у вказаній папці (рис.3.4).

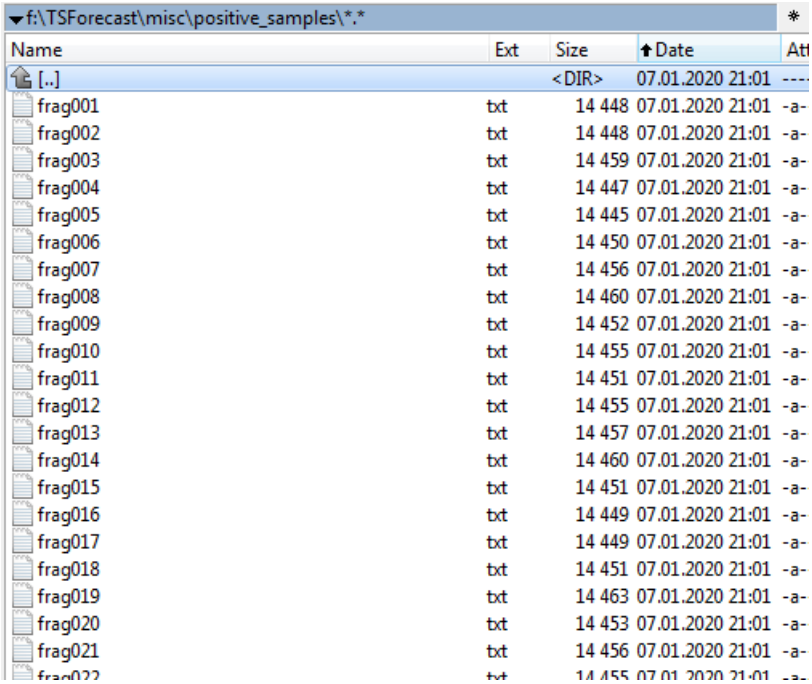


Рис.3.4. Фрагменти (зразки), збережені у вказаній папці

У разі, якщо файл або папка не обрані, ці виключення оброблюються, і користувач отримує повідомлення. Наприклад, на рис.3.5 показане повідомлення про не обрання папки.

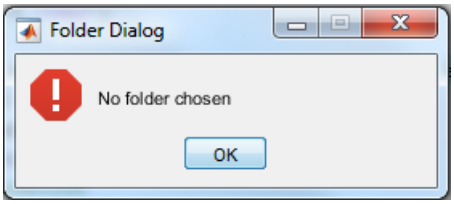


Рис.3.5. Обробка виключень при роботі з файловою системою

Нарізку доцільно виконувати для файлів з даними про штатну роботу, котрі можуть налічувати сотні тисяч вимірювань.

Якщо в системі виникає (або штучно створюється) неполадка, такий вектор $X(t)$, як правило, набагато коротший. В файлах з такими даними можна або виділити характерний фрагмент, або взяти файл в цілому в якості зразка для даної неполадки.

В результаті нарізки та накопичення записів з подіями, що відповідають певним неполадкам, для кожного з режимів на дослідній установці було отримано 100 розгорток сигналу у часі.

Наступним кроком необхідно провести згортку цих зразків у 16-векторі, групування всіх векторів та їх нормування на еталонний сигнал (див. Додаток А).

Скрипт для масової обробки сигналів (100 і більше файлів одночасно), їх згортки та нормування поданий у Додатку В.

Для початку роботи запускаємо скрипт samples2vectors.m, обираємо папку із зразками у системному вікні файлового діалогу (рис.3.6).

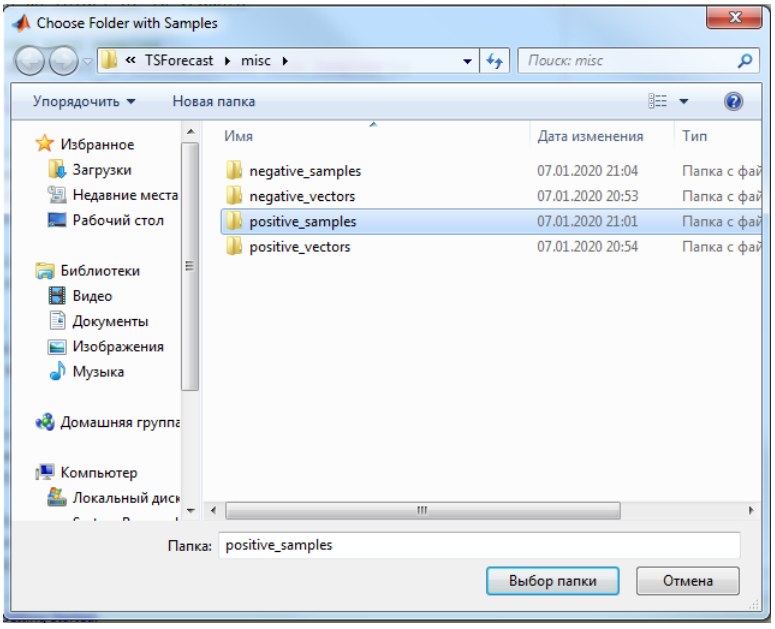


Рис.3.6. Вибір папки із зразками для отримання 16-векторів

Далі, призначаємо папку для запису векторів (рис.3.7).

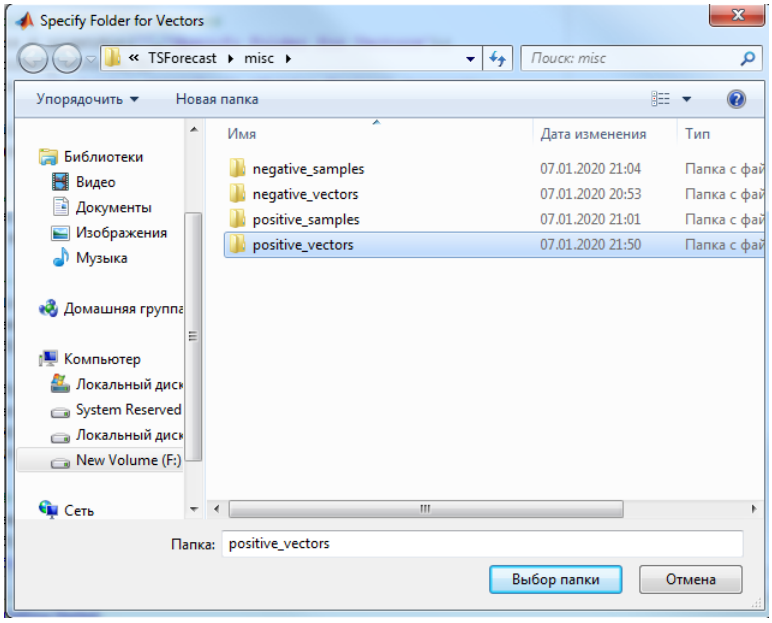


Рис.3.7. Призначення папки для запису 16-векторів

Призначаємо еталонний зразок, на який будуть нормуватися показники (рис.3.8).

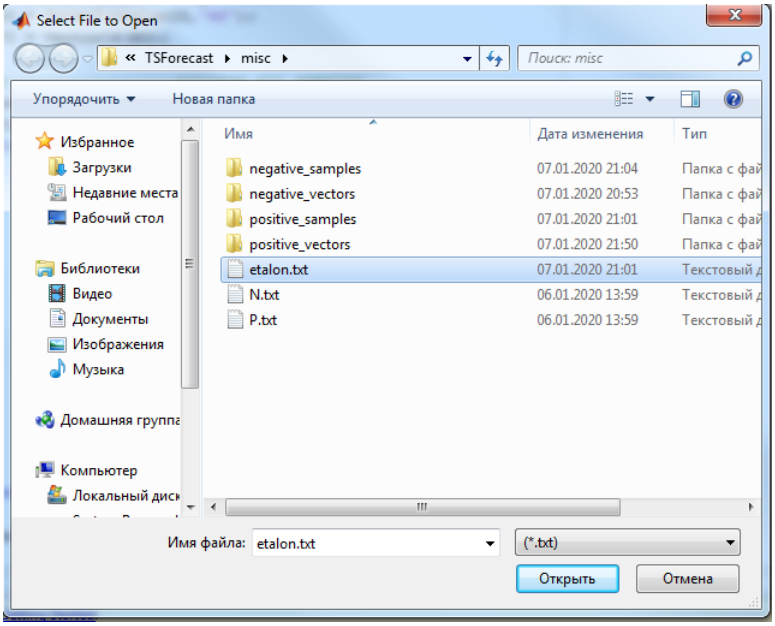


Рис.3.8. Призначення еталонного зразку для нормування

Через декілька секунд отримуємо повідомлення про успішне виконання (рис.3.9).

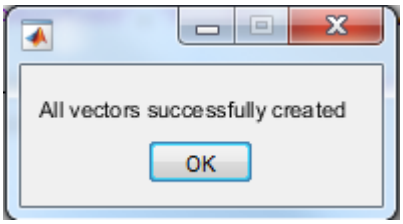


Рис.3.9. Повідомлення про успішне створення множини 16-векторів

Всі вектори отримують ім’я та нумеруються автоматично та зберігаються у вказаній папці (рис.3.10).

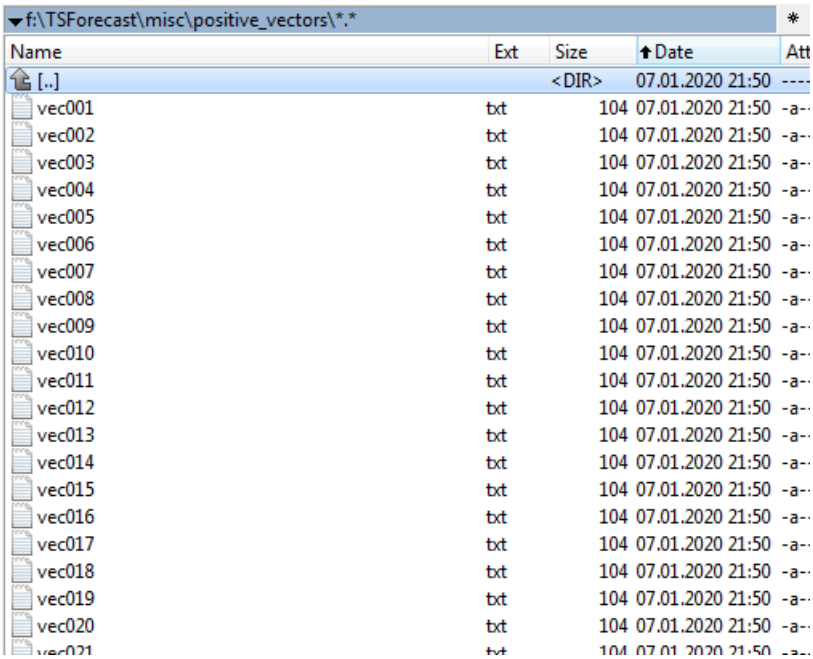
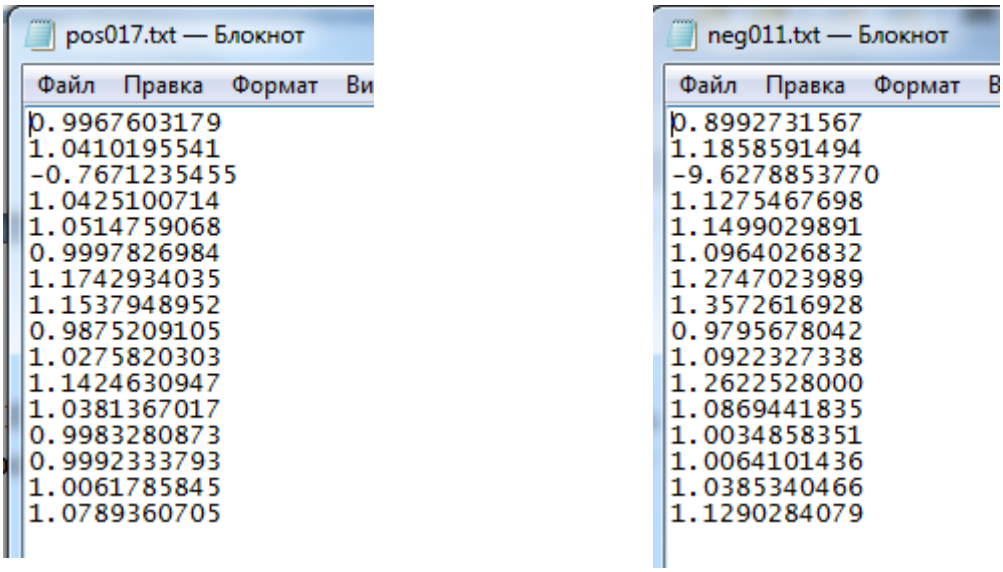


Рис.3.10. Зберігання 16-векторів у вказаній папці

У разі, якщо файл або папка не обрані, ці виключення оброблюються, і користувач отримує відповідне повідомлення (наприклад, див.рис.3.5).

На рис.3.11 представлені приклади текстових файлів із нормованими 16-векторами для позитивного (P) та негативного (N_1) зразків.

Ми бачимо, що у даному разі найбільша відмінність спостерігається для 3-го компоненту векторів – середнього відхилення, що є характеристикою саме середнього рівня шуму (див. табл.1.1). Незважаючи на нормування, значення відрізняються на порядок за абсолютною величиною.



а) Позитивний зразок (P)

б) Негативний зразок (неполадка N_1 - зростання шуму)

Рис.3.11. Нормовані 16-вектори

Подібні характерні відхилення спостерігаються і для інших видів неполадок, що вказує на ефективність запропонованої згортки.

Таким чином, тестування модуля нарізки файлів та отримання згорток у 16-вектори пройшло успішно. Нами отримано 600 згорток – по 100 на кожний із видів сигналів (рис.3.12).

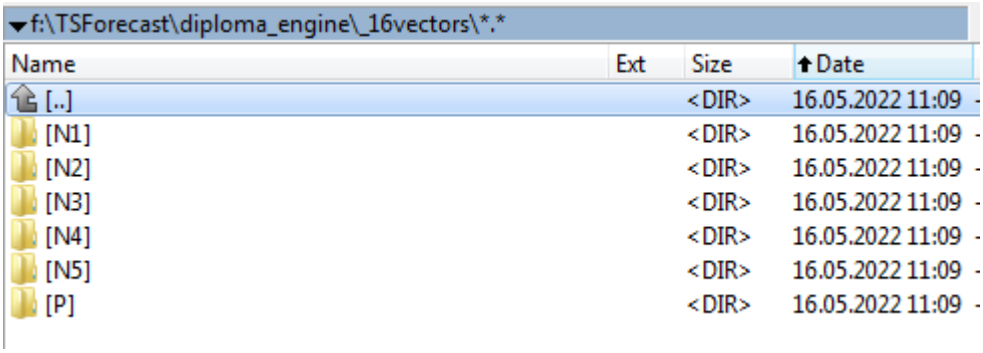


Рис.3.12. Результати Етапу 1: папки із векторами згорток

Протестуємо можливості Модуля 1 «Візуалізація та маркування» пакету TSForecast.

Для початку роботи, запускаємо на виконання TSF_Module1.exe (рис.3.13).

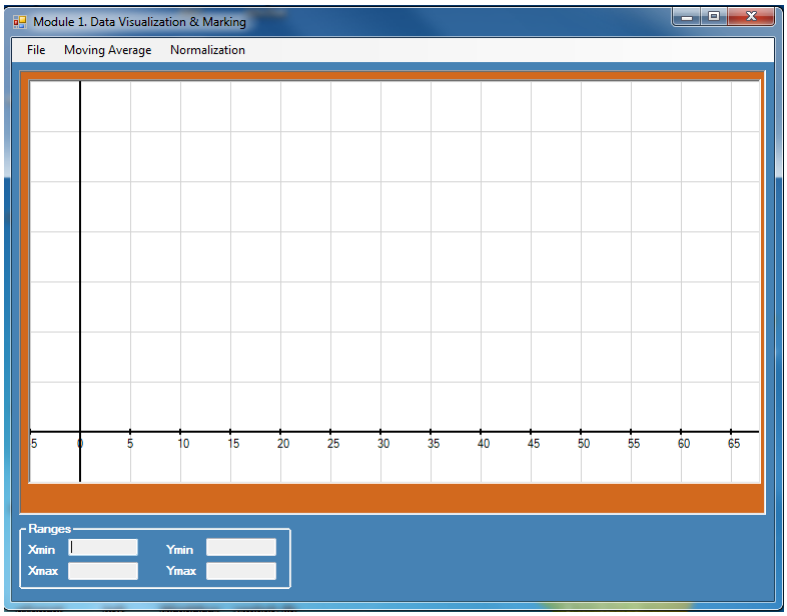


Рис. 3.13. Головне вікно Модуля 1 «Візуалізація та маркування»

Опції меню Модуля 1 показані на рис.3.14.

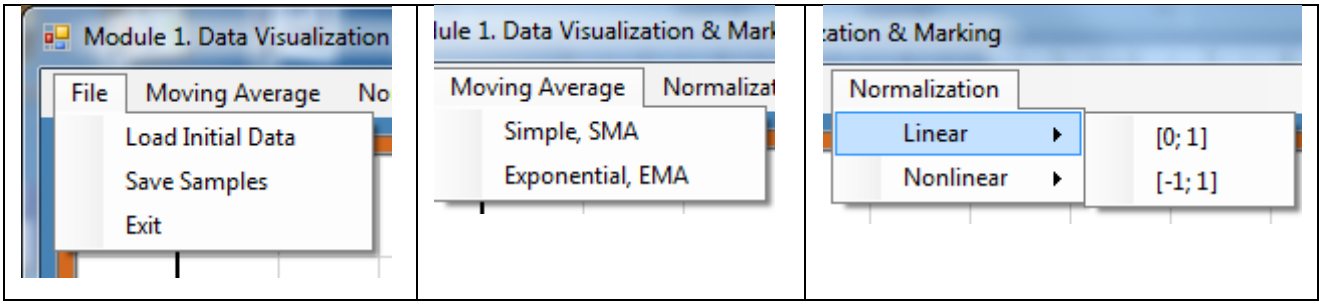


Рис.3.14. Опції меню Модуля 1

Завантаження даних відбувається у системному файловому діалозі (рис.3.15).

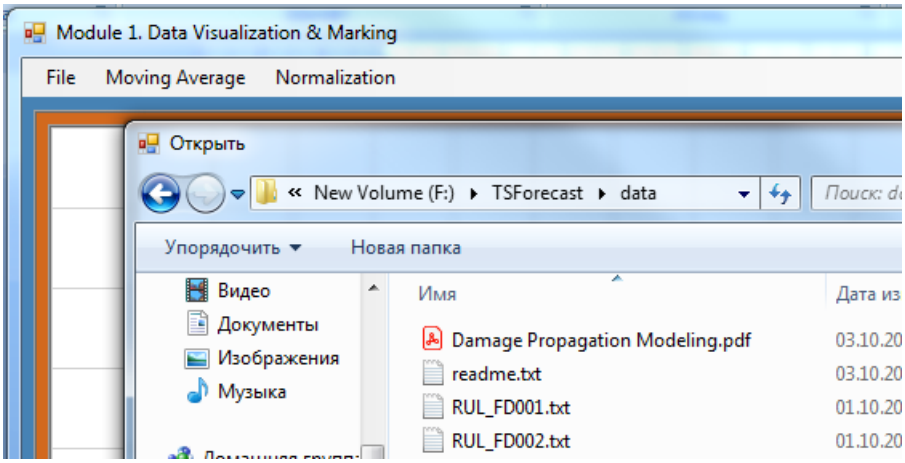


Рис.3.15. Завантаження файлу з даними

Візуалізація даних показана на рис.3.16.

Тут $X_{max} = 1071563$ - загальна кількість точок у файлі, $Y_{min} = 0$ - мінімальне значення сигналу, $Y_{max} = 9131.03$ - максимальне значення сигналу.

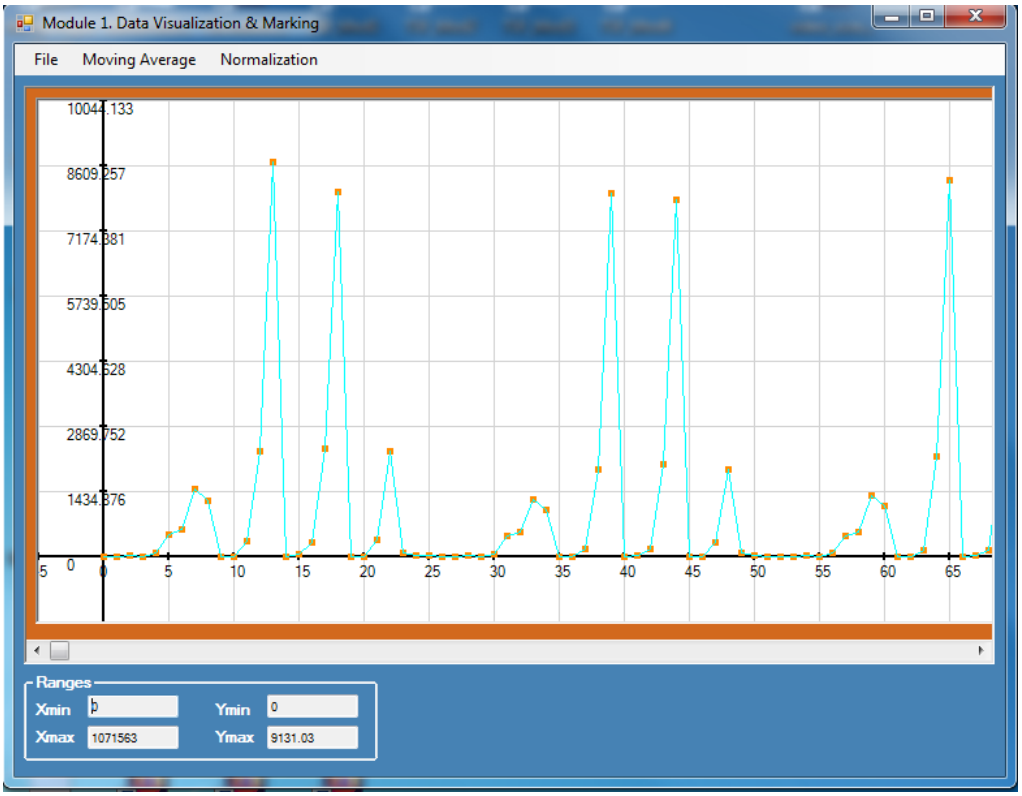


Рис.3.16. Візуалізація даних у Модулі 1

Для згладжування даних використовуються два методи (див. формули 2.1-2.2).

При виборі опції SMA, вікно згладжування задається у діалозі (рис.3.17).

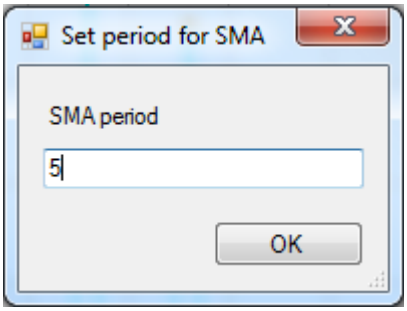


Рис.3.17. Установка значення вікна згладжування у діалозі

Результати згладжування при $m = 5$ показані на рис.3.18 і 3.19.

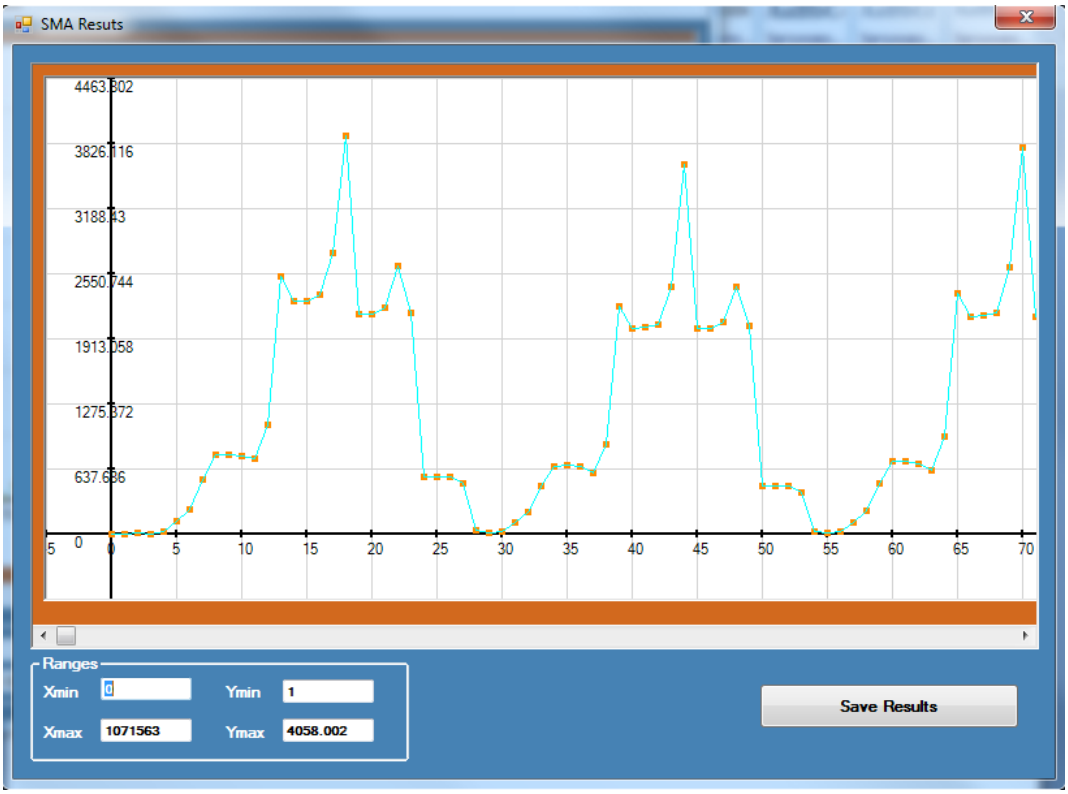


Рис.3.18. Результати лінійного згладжування за методом SMA, $m = 5$

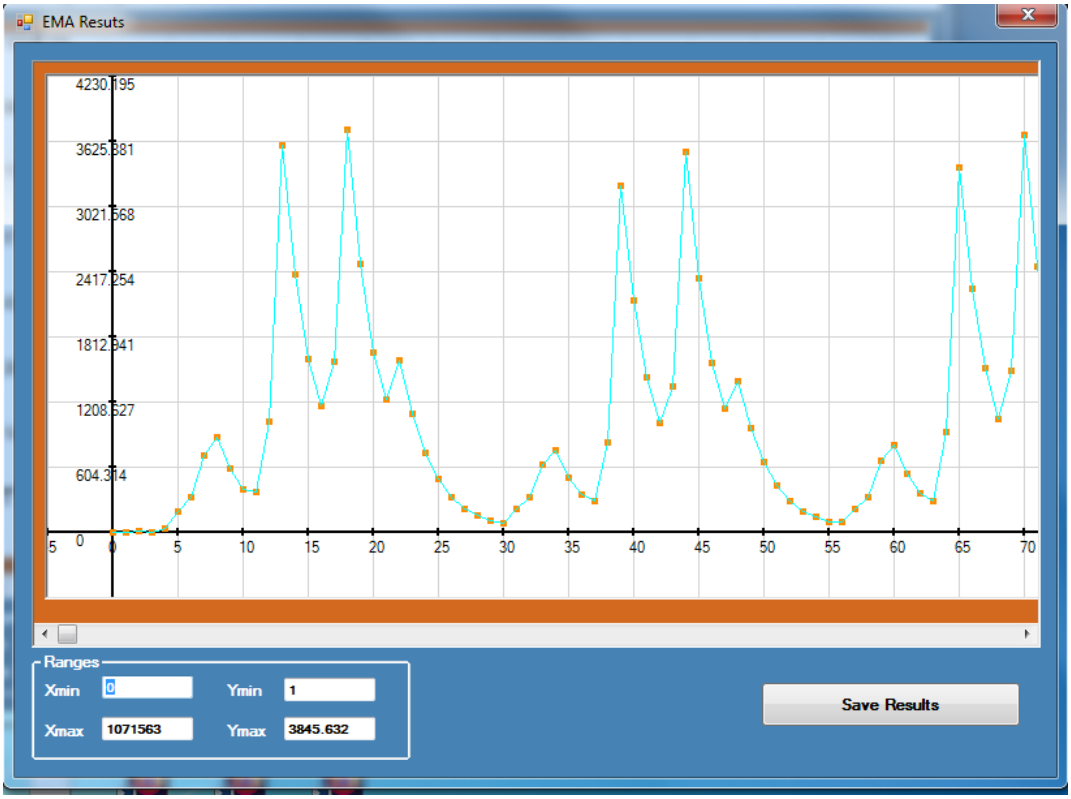


Рис.3.19. Результати експоненційного згладжування за методом EMA, $m = 5$

Нажавши кнопку Save Results, отримані результати можна запам’ятати у системному файловому діалозі.

Тестуємо можливість нормалізувати дані. Нормалізація здійснюється за формулами (2.3) – (2.6).

Результати нормалізації представлені на рис. 3.20 та 3.21.

Рис.3.20. Лінійна нормалізація у відрізок $[0; 1]$ (зверху) та $[-1; 1]$ (знизу)

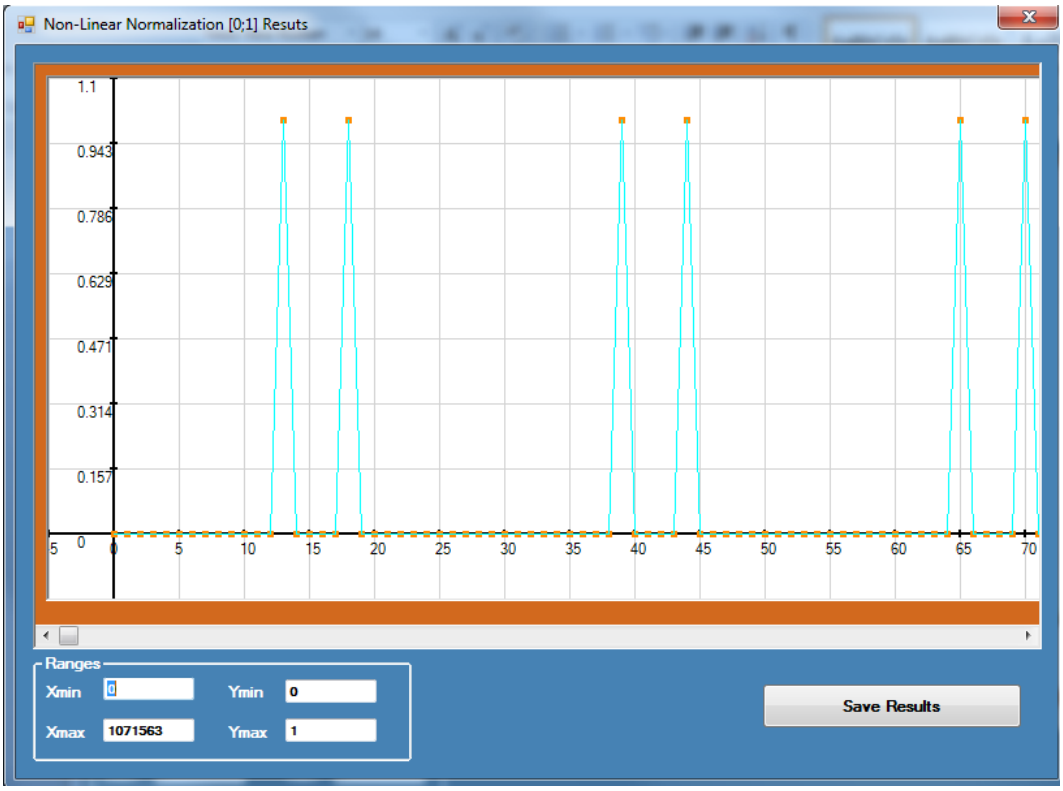


Рис.3.21. Нелінійна нормалізація у відрізок [0; 1]

Таким чином, розроблені скрипти у MATLAB та модуль 1 пакету TSForecast надає всі необхідні інструменти для обробки файлів з результатами вимірювань, отримання зразків та їх згортки у 16-вектори.

За рахунок використання паралельних розрахунків та оптимізації MATLAB при роботі з масивами та файловою системою, всі операції виконуються дуже швидко (1-2 секунди), незважаючи на значні обсяги даних ($10^5 - 10^6$ точок).

3.2. Результати навчання нейронних мереж розпізнаванню пар сигналів, отриманих при роботі двигуна у різних режимах

Для створення та навчання нейронних мереж призначено Модуль 2 пакету TSForecast. В головному вікні Модуля 2 розташоване меню з трьома розділами: File, Design, Training. Відповіді опції показані на рис.3.22.

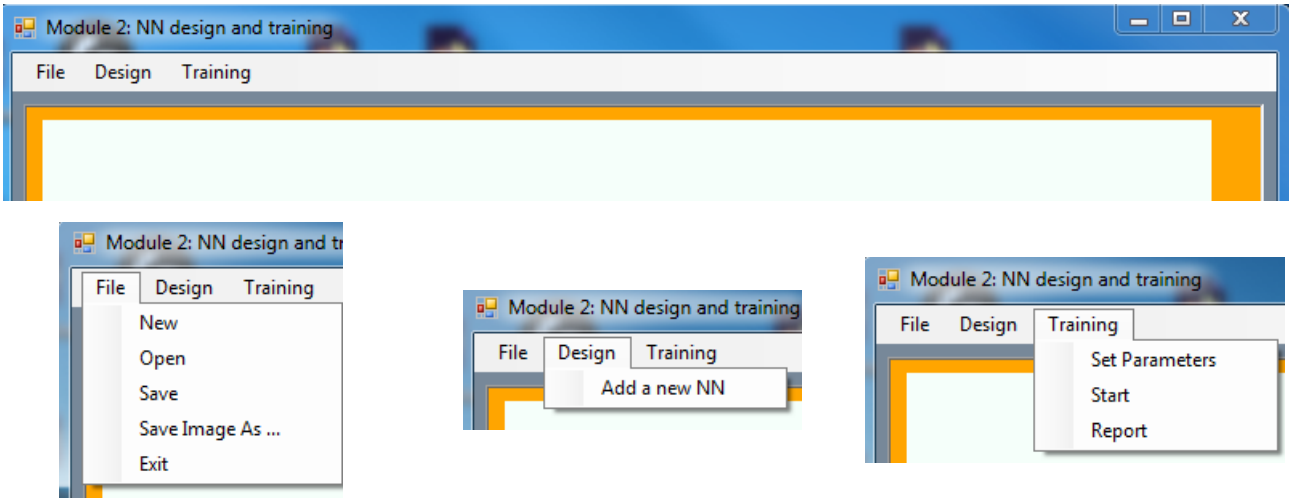


Рис.3.22. Головне меню та його опції Модуля 2 пакету TSForecast

Щоб створити нову штучну нейромережу, обираємо опцію *Design → Add a new NN*. Відкривається вікно діалогу для створення мережі (рис.3.23).

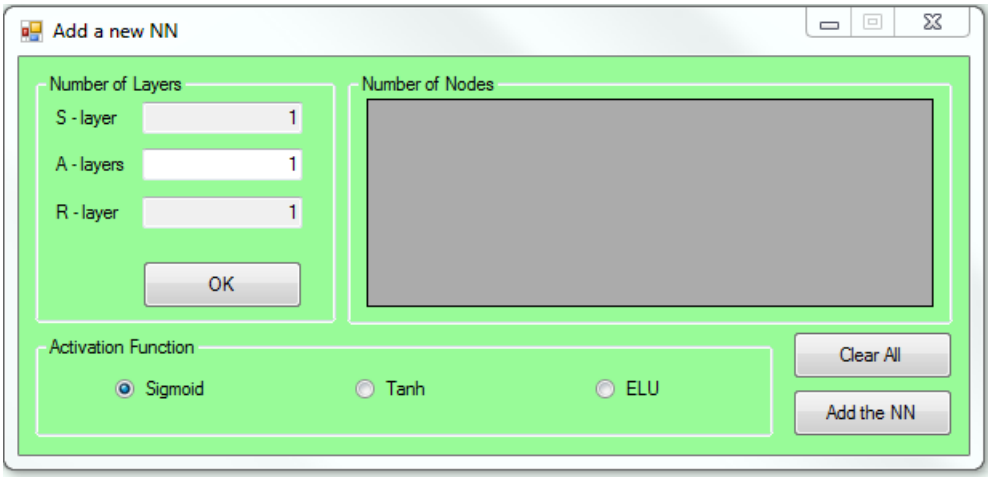


Рис.3.23. Форма для створення нової нейромережі

Текстбокси з кількістю вхідних (S-layer) та вихідних (R-layer) шарів відкриті лише для читання, оскільки може бути лише один вхідний та лише

один вихідний шар. Натомість, кількість проміжних аналітичних шарів (A-layers) можна змінювати.

Також, можна обирати функцію активації шарів. Це може біти сігмоїд, гіперболічний тангенс або ELU (див. табл.1.3).

Після введення кількості проміжних шарів та натискання кнопки ОК справа вверху активізується таблиця з запрошенням ввести кількість вузлів в кожному шарі. При цьому в вихідному шарі за замовчуванням знаходиться 2 вузли («так» та «ні») (рис.3.24).

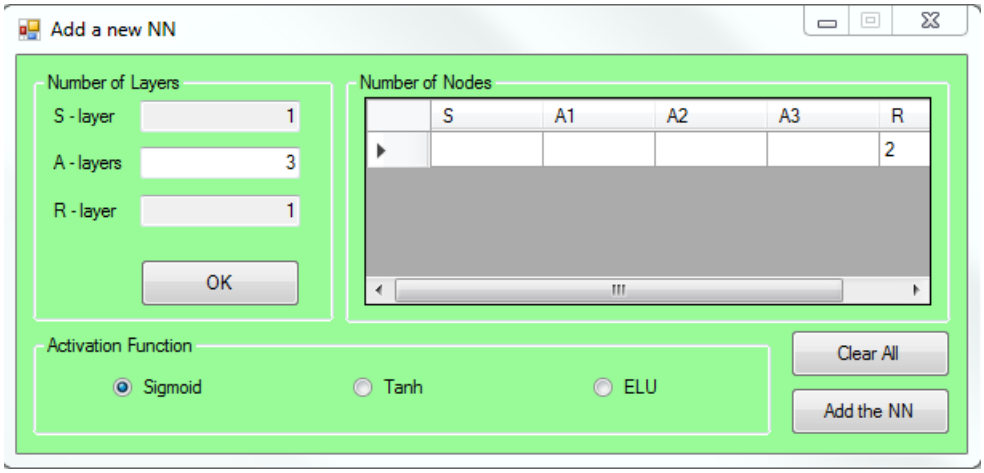


Рис.3.24. Запрошення до вводу кількості вузлів в кожному шарі

За динамічне створення таблиці відповідає метод ArrangeDG() форми fmAddANN

```
//---------------arranging DataGrid
        private void ArrangeDG()
        {
            int NC = H + 2; //number of columns
            DataGridViewColumn[] DGcolumns = new DataGridViewColumn[NC];

            DGcolumns[0] = new DataGridViewColumn();
            DGcolumns[0].HeaderText = "S";
            DGcolumns[0].Name = "S";
            for (int i = 1; i<NC-1;i++)
            {
                DGcolumns[i] = new DataGridViewColumn();
                DGcolumns[i].HeaderText = "A" + i.ToString();
                DGcolumns[i].Name = "A" + i.ToString();
            }
            DGcolumns[NC - 1] = new DataGridViewColumn();
            DGcolumns[NC - 1].HeaderText = "R";
            DGcolumns[NC - 1].Name = "R";

            for (int i = 0; i < NC; i++)
            {
                DGcolumns[i].Width = 70;
```

```
            DGcolumns[i].CellTemplate = new DataGridViewTextBoxCell();
            dataGridView1.Columns.Add(DGcolumns[i]);
        }
        dataGridView1.Rows.Add();
        dataGridView1.Rows[0].Cells[NC - 1].Value = "2";
        dataGridView1.Rows[0].Cells[NC - 1].ReadOnly = true;
        dataGridView1.AllowUserToAddRows = false;
        dataGridView1.CurrentCell = dataGridView1.Rows[0].Cells[0]; //on focus
        dataGridView1.BeginEdit(true);
    }
```

Вводимо кількість вузлів (рис.3.25).

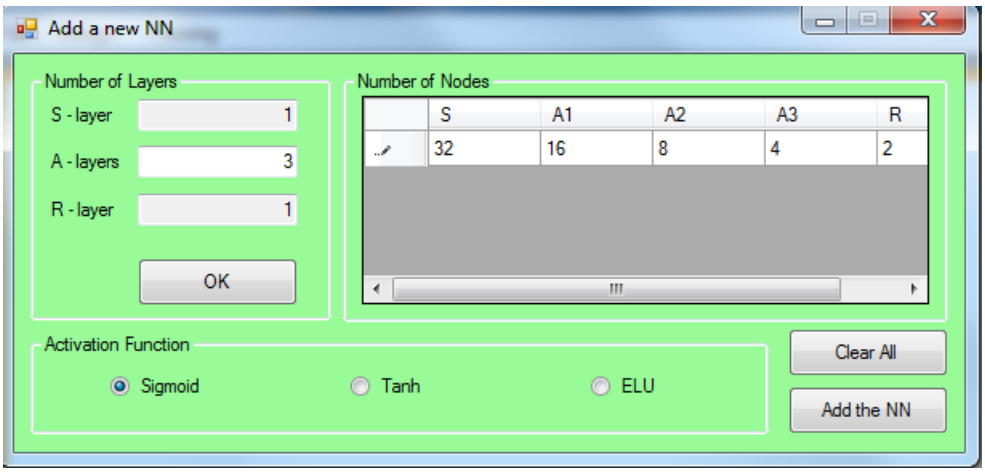


Рис.3.25. Архітектура нейромережі задана

Після натискання кнопки Add the NN отримуємо зображення нейромережі з можливістю її тренування (рис.3.26).

На зображенні сині лінії відповідають зв'язкам з від'ємними вагами $w_{ij} \leq -0.005$, жовтогарячі лінії – зв'язкам з додатними вагами $w_{ij} \geq 0.005$. Якщо під час навчання ваги зменшаться до $|w_{ij}| < 0.005$, вони будуть зображуватися лініями сірого кольору.

Зображення генерується методом PaintANN(NeuralNet nn) головної форми Form1.

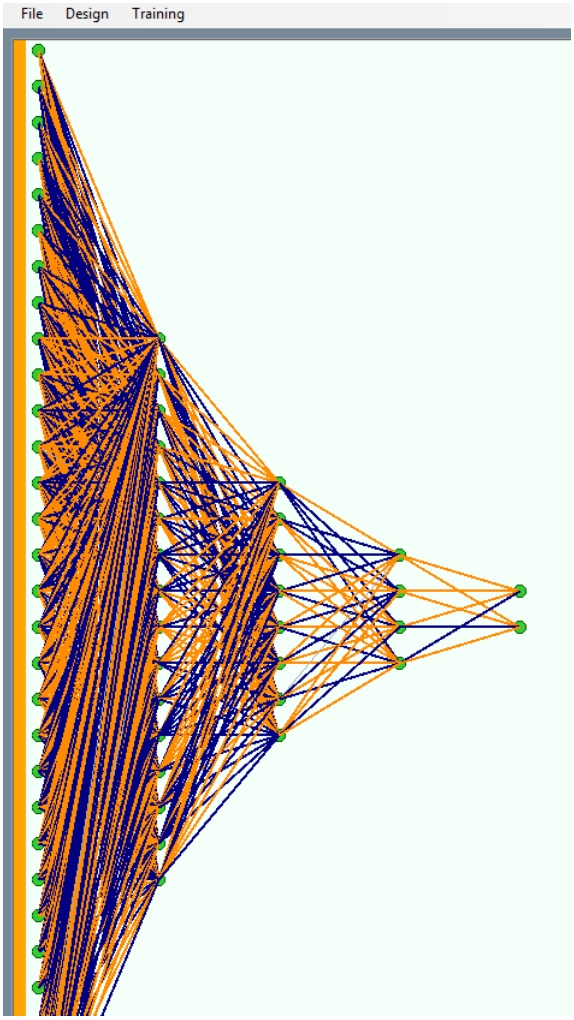


Рис.3.26. Зображення нейромережі.

Обираємо опцію *Training → Set Parameters*. Відкривається діалогове вікно з формою для визначення параметрів навчання (рис.3.27)

У блоці Work Folders за допомогою системного діалогу встановлюємо шляхи до папок з позитивними та негативними зразками. При навчанні вихід нейромережі «так» буде порівнюватися з позитивними зразками, а вихід «ні» - з негативними.

Навчання може проводитися в одному із трьох режимів (Tactics of Gradient Descent): стохастичному, упорядкованому або змішаному (детальніше - див. розділ 2.2).

В формі також необхідно заповнити поля із загальною кількістю епох, швидкістю навчання η та моменту навчання α (див. формулу 1.9).

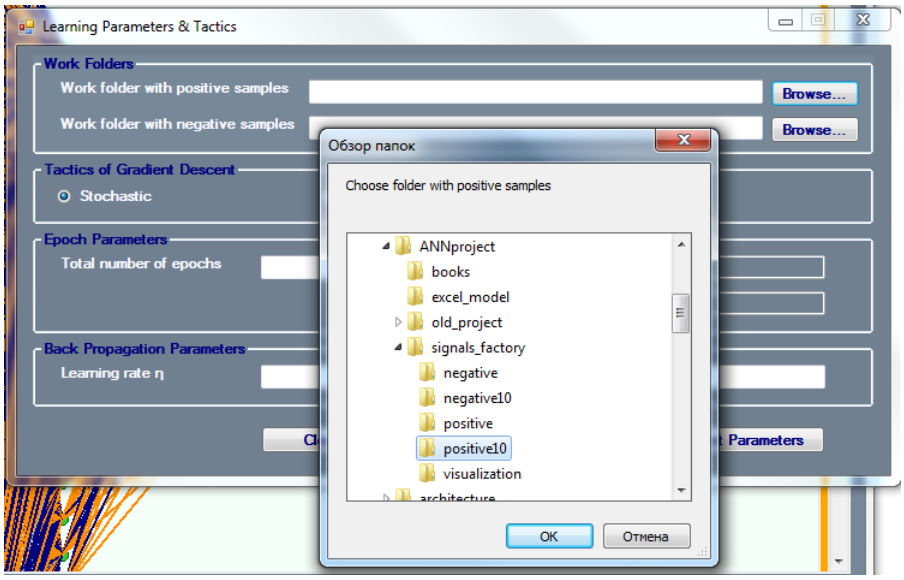


Рис.3.27. Форма для визначення параметрів навчання, системний діалог для пошуку папок із зразками

Після визначення всіх параметрів, нажимаємо кнопку *Set Parameters* та отримуємо повідомлення про успішне встановлення (рис.3.28).

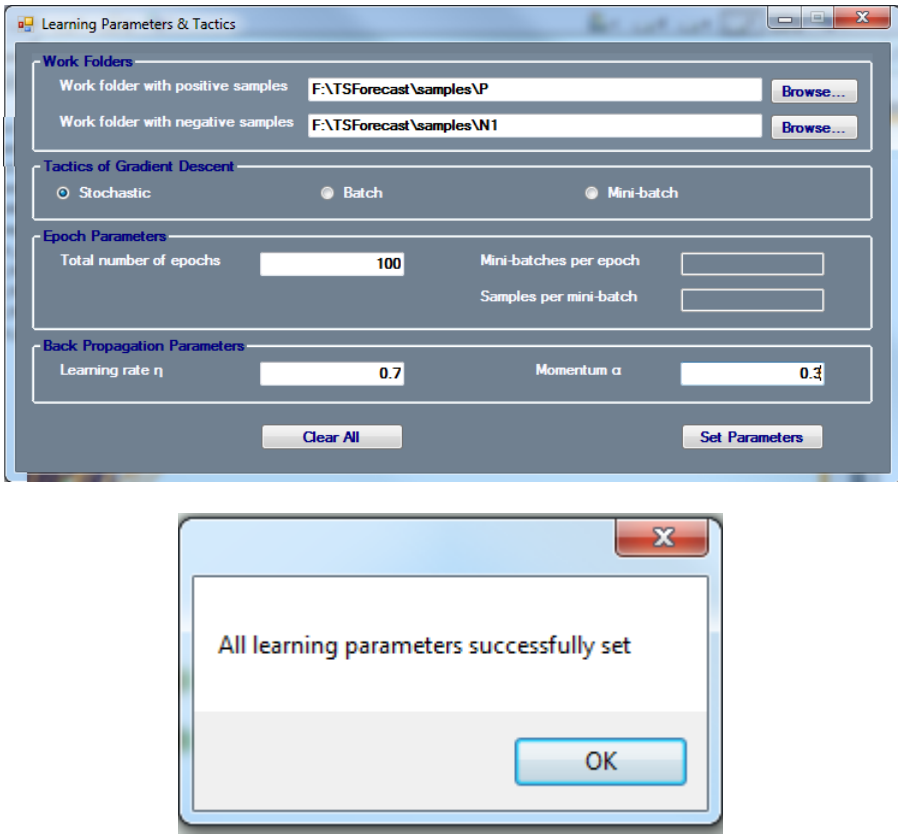


Рис.3.28. Параметри навчання успішно встановлено

В головному меню обираємо опцію Training → Start та переходимо до навчання.

Процес навчання займатиме деякий час, отже, в асинхронному режимі для користувача зявляється інформаційне вікно зі статус-баром про хід процесу (рис.3.29).

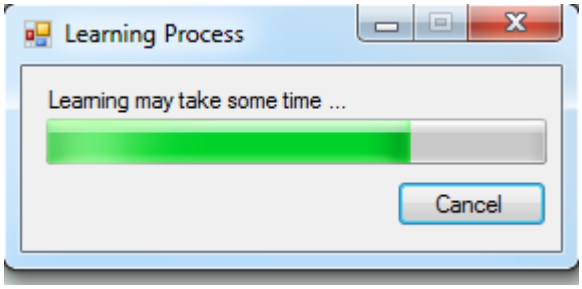


Рис.3.29. Інформаційне вікно зі статус-баром про хід навчання

За асинхронну роботу форми fmLearningProcess відповідають методи

```
private void backgroundWorker1_DoWork(object sender, DoWorkEventArgs e)
private void backgroundWorker1_ProgressChanged(object sender,
ProgressChangedEventArgs e)
private void backgroundWorker1_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
```

Коли навчання закінчується, з’являється повідомлення (рис.3.30).

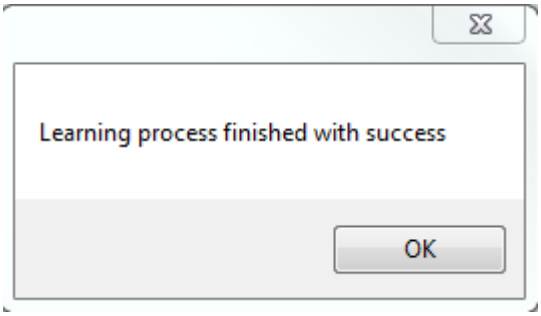


Рис.3.30. Повідомлення про закінчення навчання

Після натиснення ОК, отримуємо зображення нейромережі на початку та в кінці навчання (рис.3.31).

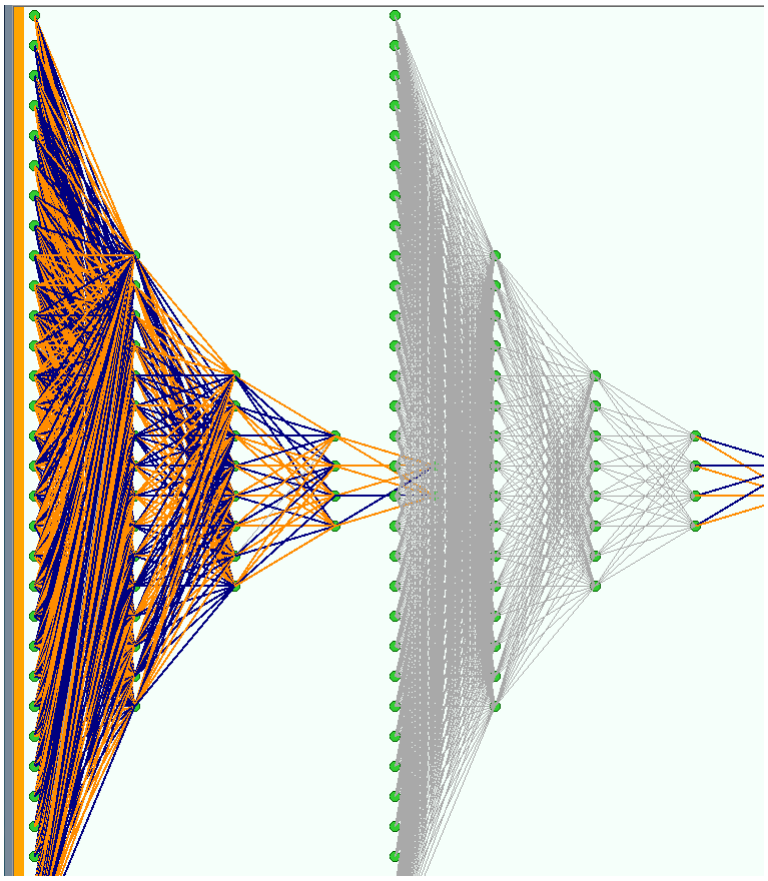


Рис.3.31. Початкова нейромережа (зліва) та кінцева нейромережа (справа)

Обираємо опцію *Training* → *Report* та отримуємо форму зі звітом про хід навчання (рис.3.32).

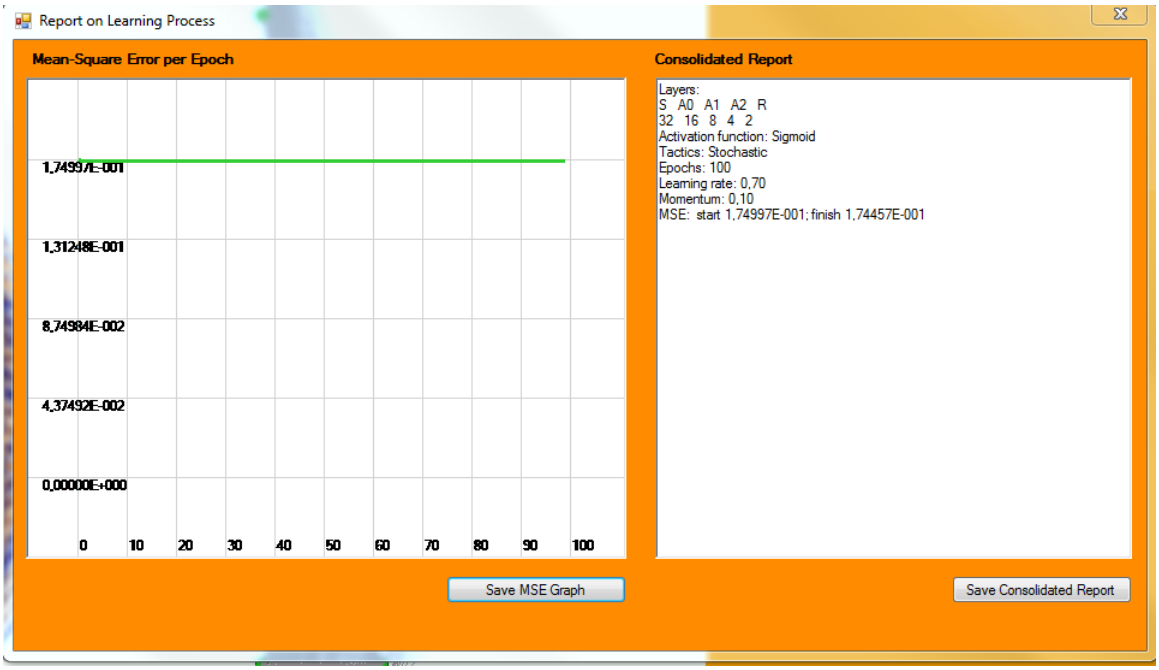


Рис.3.32. Звіт про хід навчання (навчання не відбулося)

Ця форма є інструментом для первинного візуального та розрахункового контролю успіху навчання.

В даному випадку, за видом графіка зміни середнього квадратичного відхилення з кожною епохою, можемо зробити висновок, що навчання не відбулося. СКВ повинно зменшуватися, а цього майже не відбувається.

В такому випадку рекомендується:

– змінити архітектуру мережі (змінити кількість проміжних шарів, кількість вузлів у шарах, та/або функцію активації вузлів);
– збільшити параметри навчання (кількість епох, швидкість навчання η та/або момент α).

Проведемо ще один експеримент. Зменшимо кількість шарів, кількість вузлів, оберемо тангенційну функцію активації (рис. 3.33).

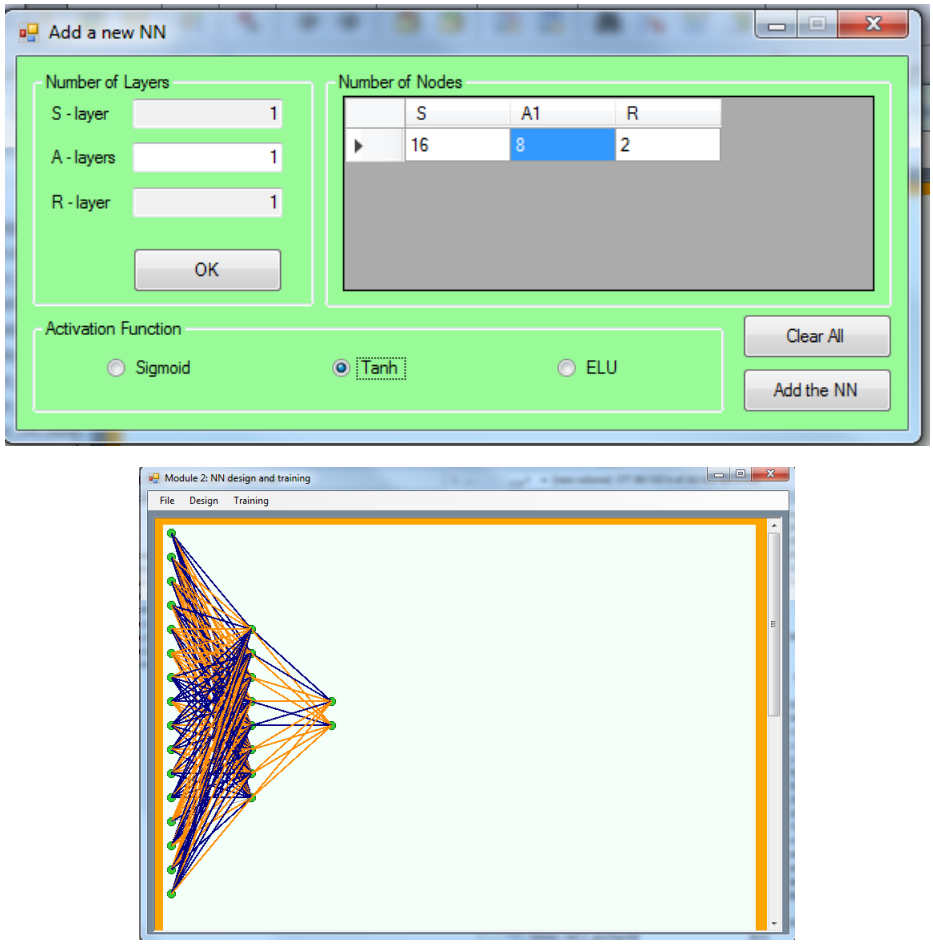


Рис.3.33. Зміна архітектури нейромережі

Змінимо параметри навчання (рис.3.34).

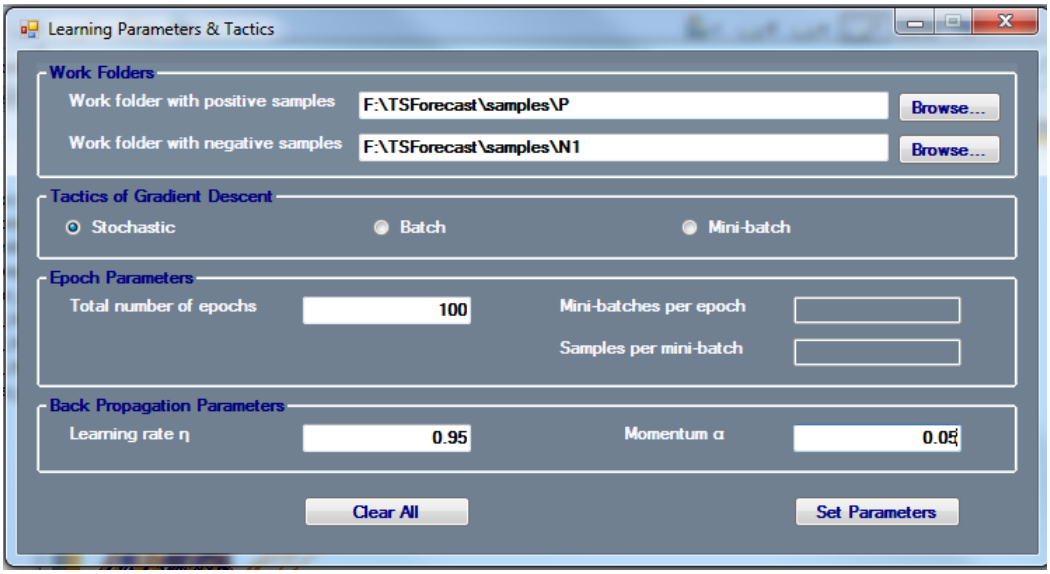


Рис.3.34. Зміна параметрів навчання

Отримуємо не таке «сіре» зображення для кінцевої нейромережі (рис.3.35).

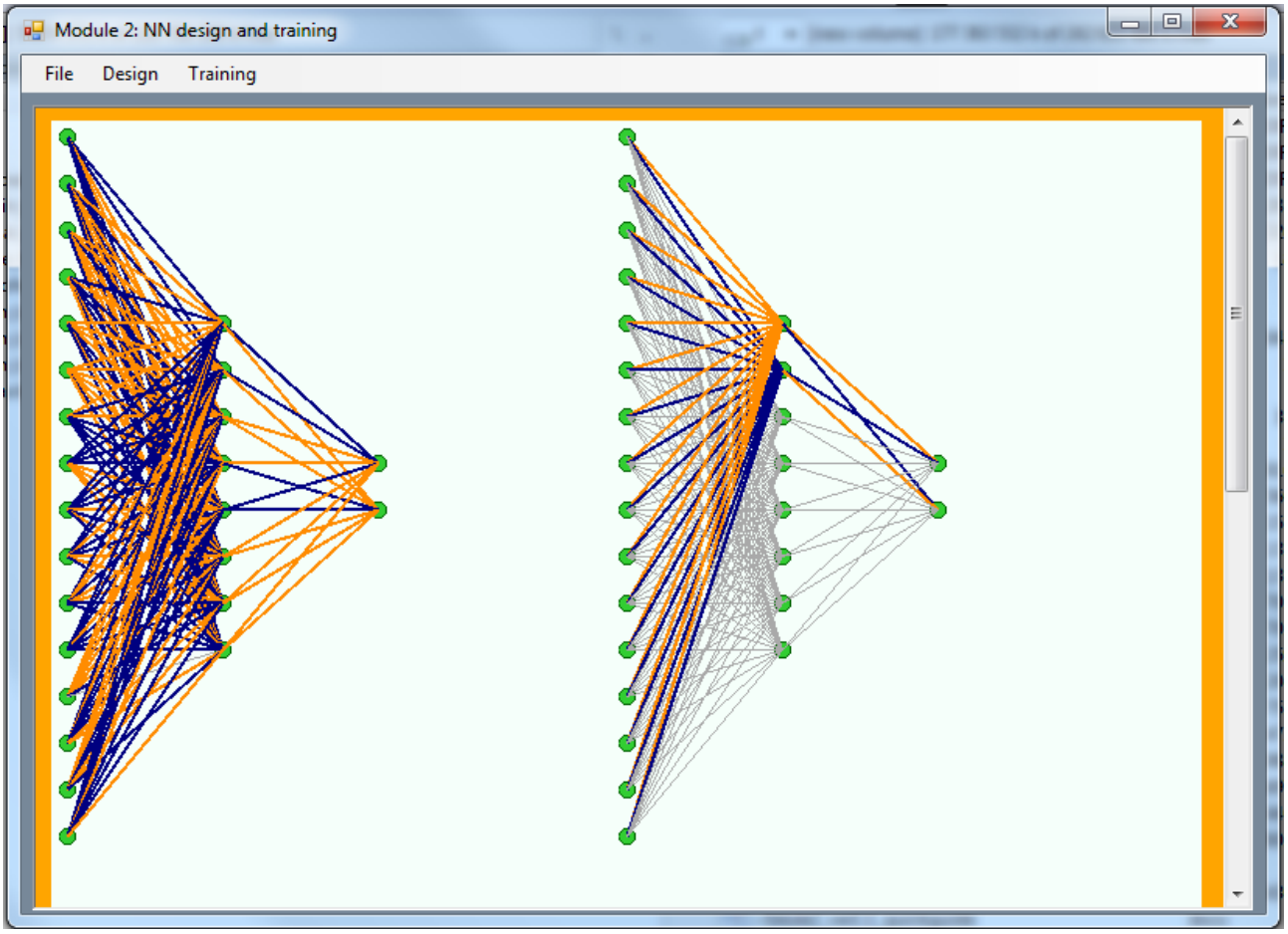


Рис.3.35. Кінцева нейромережа має не тільки сірі, але й кольорові зв’язки

Така зміна кольорів вказує на навчення мережі, оскільки показує, що певна кількість зв’язків є значимою, з $|w_{ij}| \geq 0.005$.

Дійсно, згідно звіту отримуємо різке зниження СКВ вже після другої епохи (рис.3.36), що є ознакою навчання.

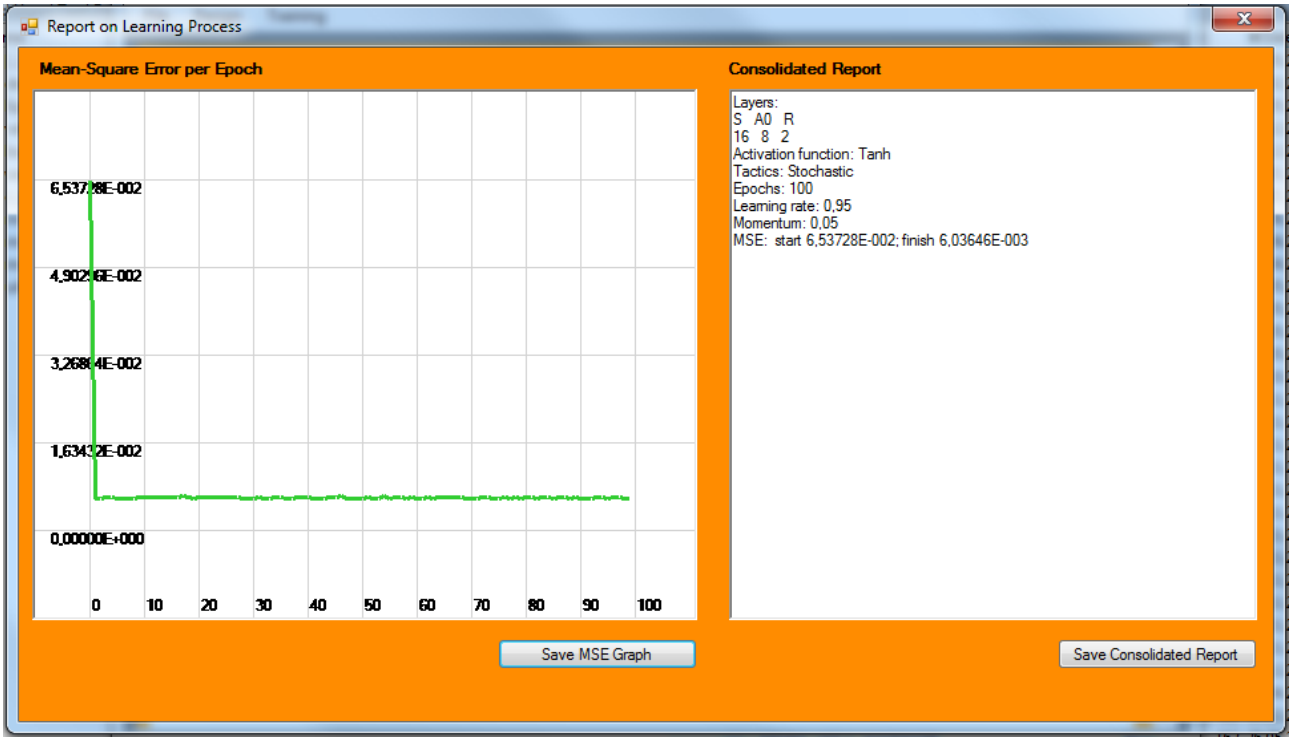


Рис.2.36. Звіт із успішного навчання

Із звіту можна запам’ятати як отриманий графік (зліва), так і консолідований звіт (справа). Обидва звіти зберігаються у текстових файлах (рис.2.37).

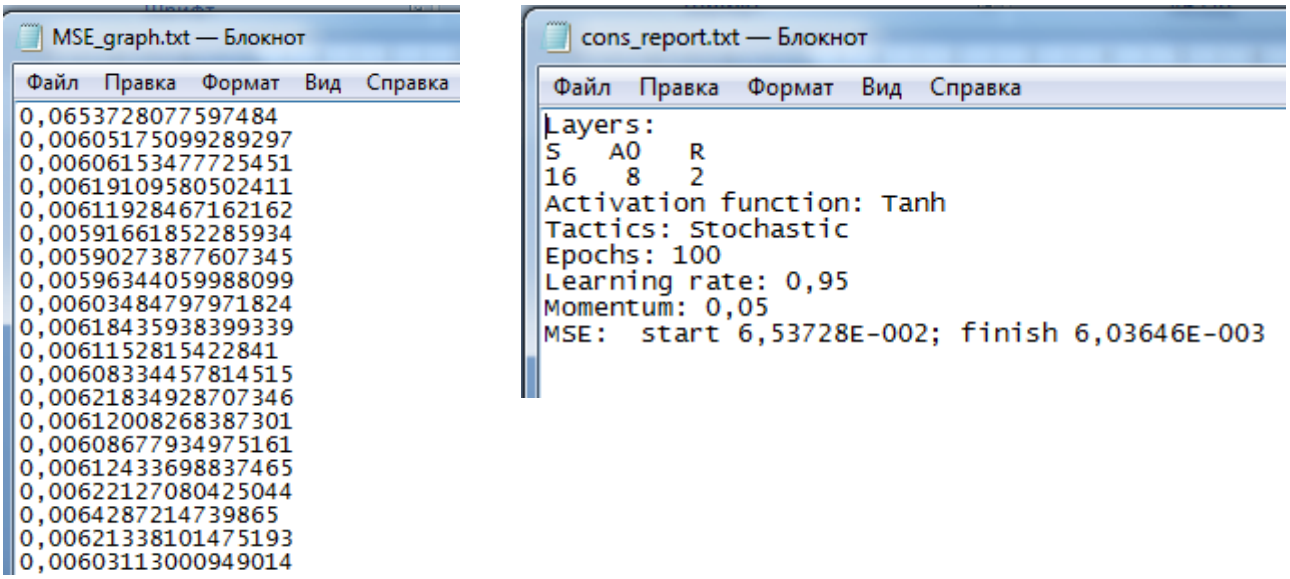


Рис.3.37. Зберігання точок графіку та тексту звіту у текстових файлах

Навчена нейромережа зберігається у бінарному файлі після вибору опції *File → Save* (рис.3.38). Для цього у класі NeuralNet реалізовано інтерфейс ISerializable (див. Розділ 2.2)

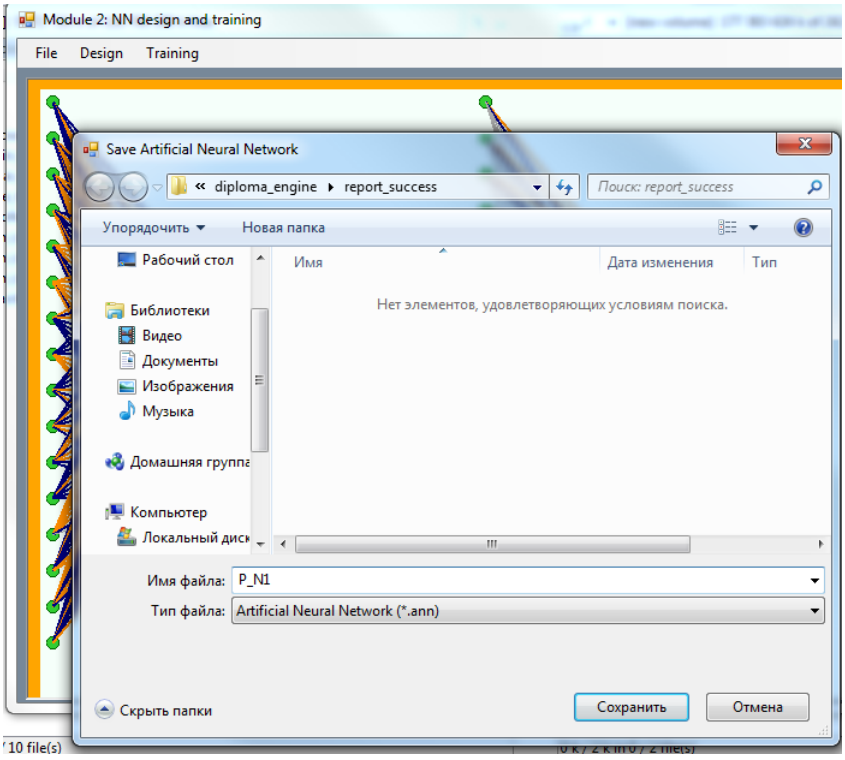


Рис.3.38. Зберігання нейромережі у бінарному файлі

Також, у Модулі 2 є опція збереження зображення нейромережі до та після навчання *File → Save Image As…* Доступні три формати зображень: bmp, jpg та png (рис.3.39) Зберігання виконується методом saveAsToolStripMenuItem_Click головної форми Form1

```
//-------------------------------------Save Image
        private void saveAsToolStripMenuItem_Click(object sender, EventArgs e)
        {
            SaveFileDialog save_image = new SaveFileDialog();
            save_image.Filter = "Bitmaps|*.bmp|PNG files|*.png|JPEG files|*.jpg";
            save_image.Title = "Save image of network/networks";
            ImageFormat format = ImageFormat.Jpeg;

            if (save_image.ShowDialog() == DialogResult.OK)
            {
                switch(save_image.Filter)
                {
                    case "*.bmp":
                        format = ImageFormat.Bmp;
                        break;
                    case "*.png":
                        format = ImageFormat.Png;
                        break;
                }

                //-----------white background
                Bitmap white = new Bitmap(canvas.Width, canvas.Height);
                using (g = Graphics.FromImage(white))
                {
                    g.Clear(Color.White);
                    g.DrawImage(canvas, 0, 0);
                }

                white.Save(save_image.FileName, format);
            }
        }
```

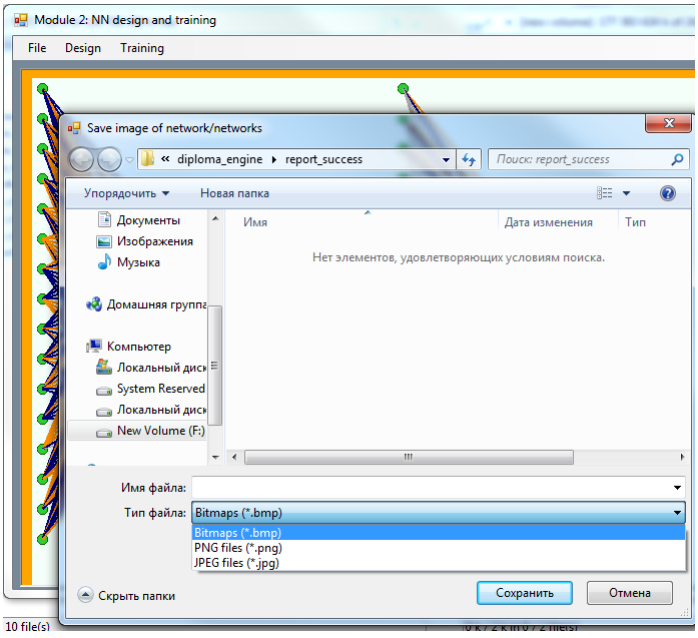


Рис.3.39. Зберігання зображення (доступні три формати)

Збережені файли з результатами навчання представлені на рис.3.40.

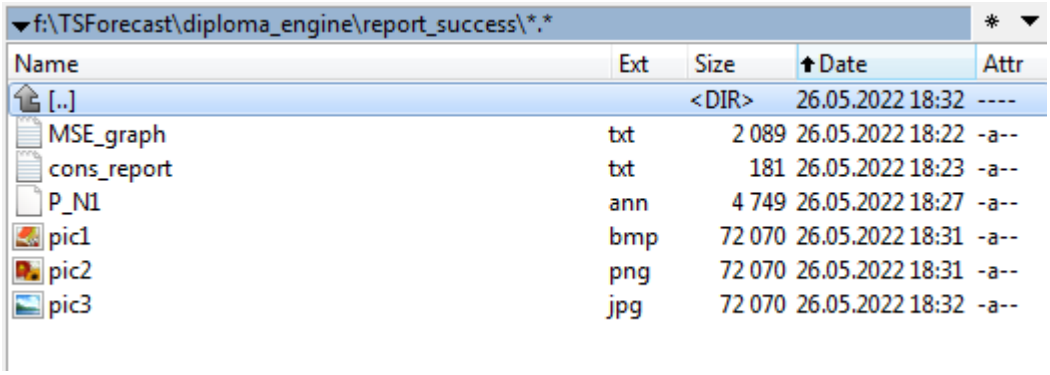


Рис.3.40. Файли з результатами навчання

Таким чином, Модуль 2 пакету TSForecast надає всі необхідні інструменти для навчання та первинного контролю його результатів. Інтерфейс додатку є зручним для користування і не потребує багато часу для вивчення та початку роботи.

Для цілей дослідження із застосуванням Модуля 2 ми навчили та відібрали 45 нейромереж (по 3 нейромережі для кожної пари сигналів) (рис.3.41).

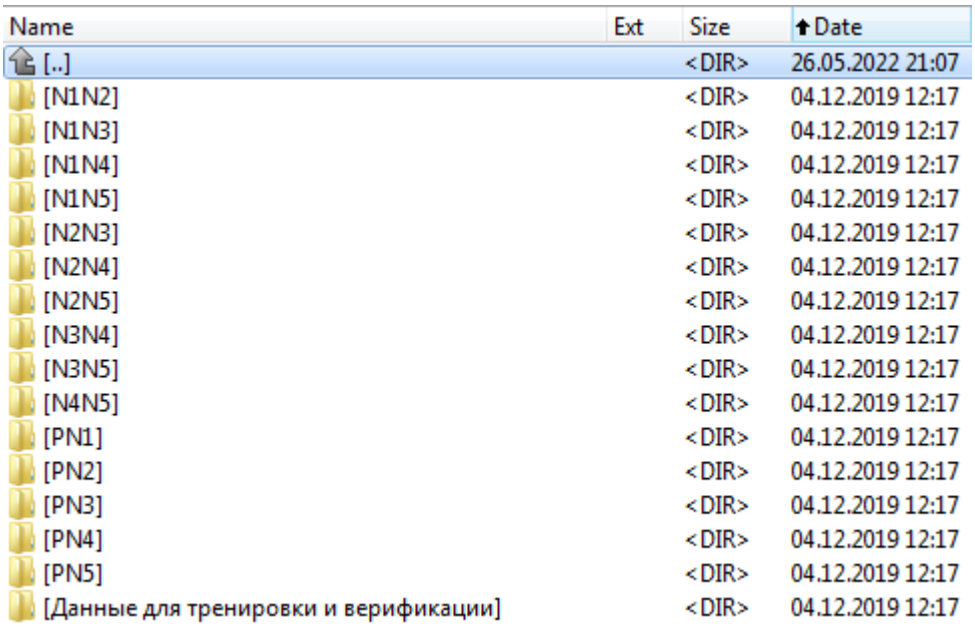


Рис.3.41. Папка із навченими нейромережами

К кожній папці з мережею зберігається сама мережа, звіти із процесу навчання та звіт верифікації (наступний етап).

3.3. Результати верифікації, відбір навчених нейронних мереж

Для процесу верифікації у пакеті TSForecast розроблено Модуль 3.

Головна форма додатку має два розділи: вхідні дані (Input Data) та результати верифікації (Verification Output) (рис.3.42).

Рис.3.42. Головна форма Модуля 3 пакету TSForecast

Вхідні дані складаються із

- бінарного файлу із навченою нейромережею;
- папки з позитивними зразками, новими для нейромережі;
- папки з негативними зразками, новими для нейромережі.

Результати верифікації нейромережі, що навчена розпізнавати штатну роботу (P - позитивний сигнал) та неполадку зростання шуму (N1 – негативний сигнал) показано на рис. 3.43. Точність складає ACC = 1 (100%).

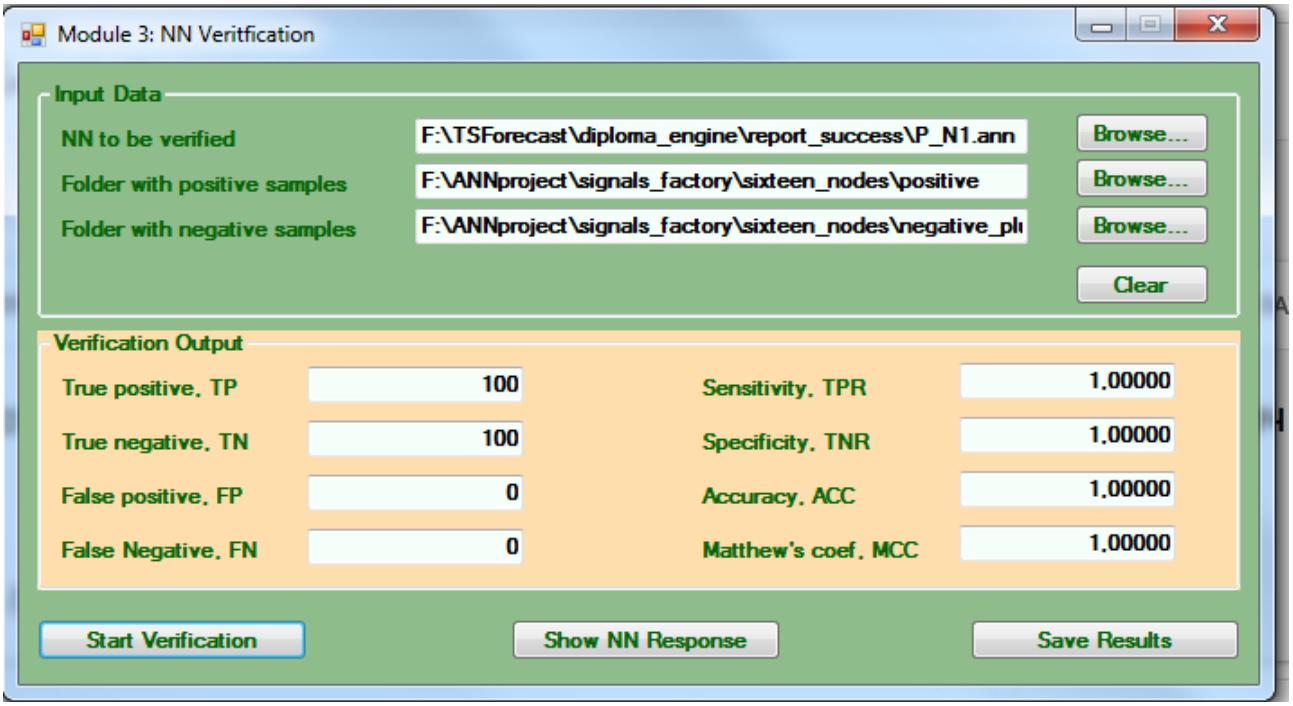


Рис.3.43. Нейромережа відмінно розпізнає сигнали P і N1

Натискаючи кнопку ShowNNResponse, ми отримуємо звіт із вихідними сигналами нейромережі (виходи «так» та «ні» у R-шарі).

Сигнали на виході збираються у списки

```
List<string> pos_response;  //info on NN responses
List<string> neg_response;
```

Використовуючи значення у вузлах вихідного шару обєкта класу NeuralNet

```
pos_response.Add(S_out[K - 1][0, 0].ToString());
pos_response.Add(S_out[K - 1][0, 1].ToString());
neg_response.Add(S_out[K - 1][0, 0].ToString());
neg_response.Add(S_out[K - 1][0, 1].ToString());
```

Далі списки передаються у дочірню форму fmResponse та відображаються у річбоксах

```
public fm_Response(List<string> pos_response, List<string> neg_response)
        {
            InitializeComponent();
            richTextBox1.Lines = pos_response.ToArray();
            richTextBox2.Lines = neg_response.ToArray();
        }
```

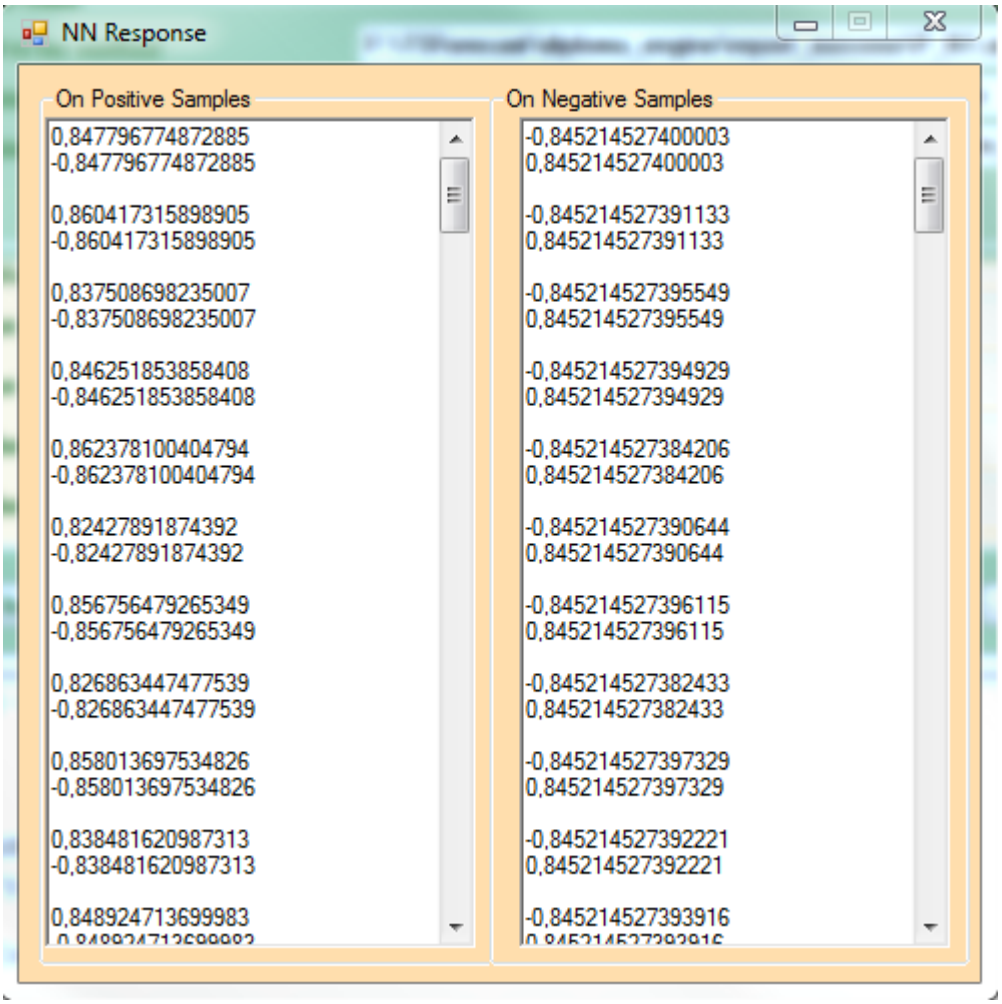


Рис.3.44. Сигнали вихідного шару нейромережі
при реакції на зразки, що надходять на її вхід в процесі верифікації

Зберегти звіт з результатами верифікації можна, натиснувши кнопку Save Results.

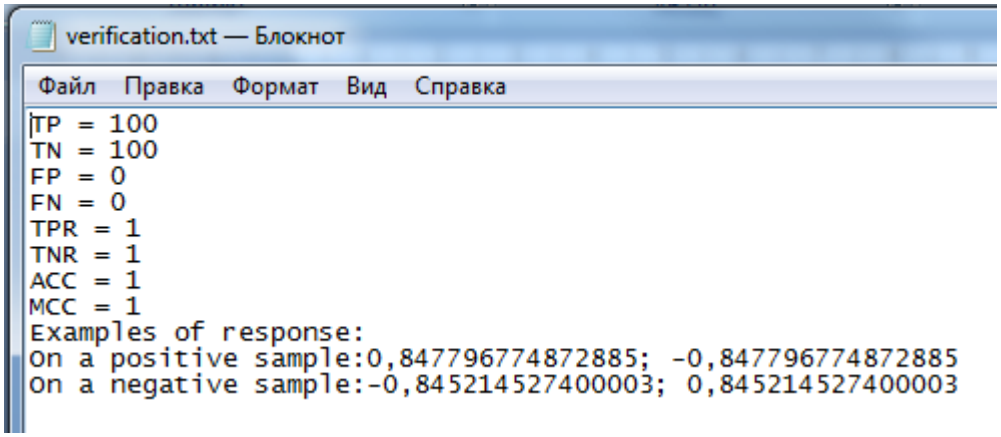


Рис.3.45. Звіт з результатами верифікації

Крім розрахованих значень показників, у звіті також зберігається перша пара відповідей вихідного шару на позитивний та негативний сигнали. Ця інформація є корисною для аналізу якості навчання.

Щоб отримати приклад з негативною реакцією, подамо на ту ж нейромережу пару інших сигналів: штатну роботу (P - позитивний сигнал) та неполадку зростання амплітуди (N2 – негативний сигнал). Як і очікується, мережа неправильно розпізнає сигнали, вважаючи, що всі сигнали N2 – також позитивні сигнали (FP = 100) (рис.3.45)

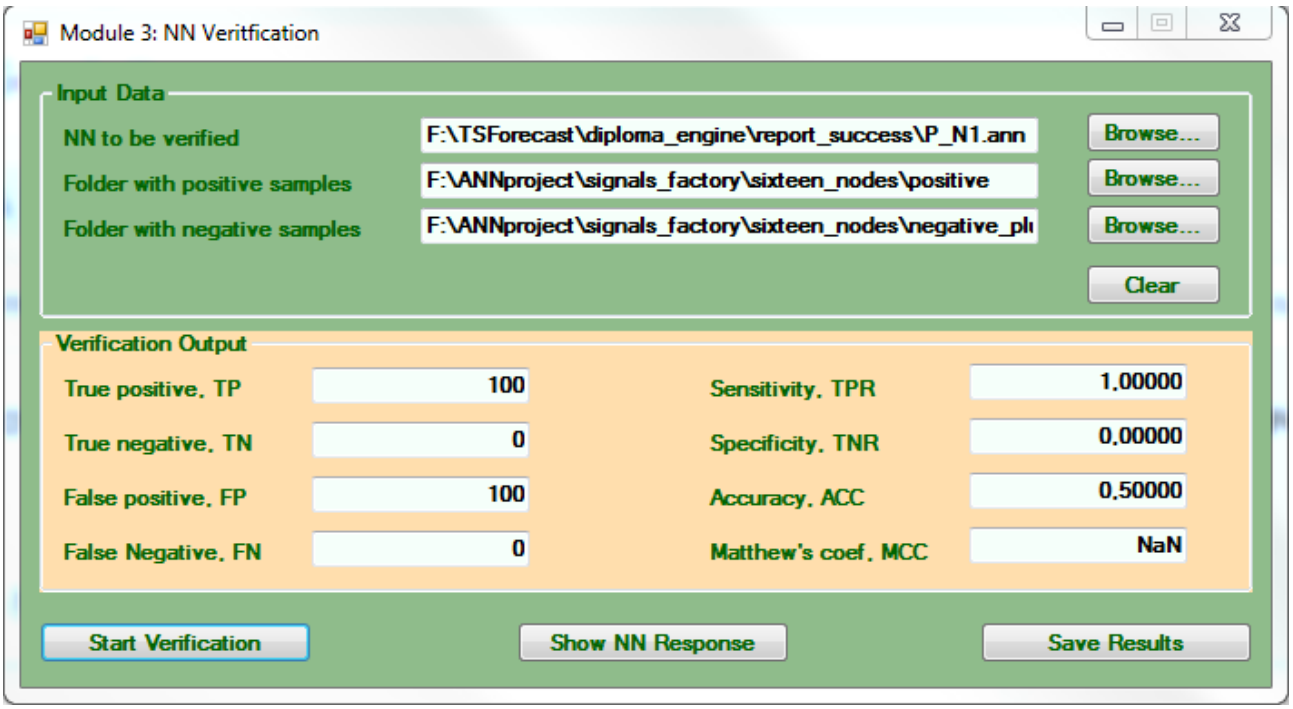


Рис.3.45. Нейромережа, навчена на сигналах P і N1,
не може розпізнати N2 із пари P і N2

Вихідний сигнал на рис.3.46 показує, що мережа однаково реагує га P і N2

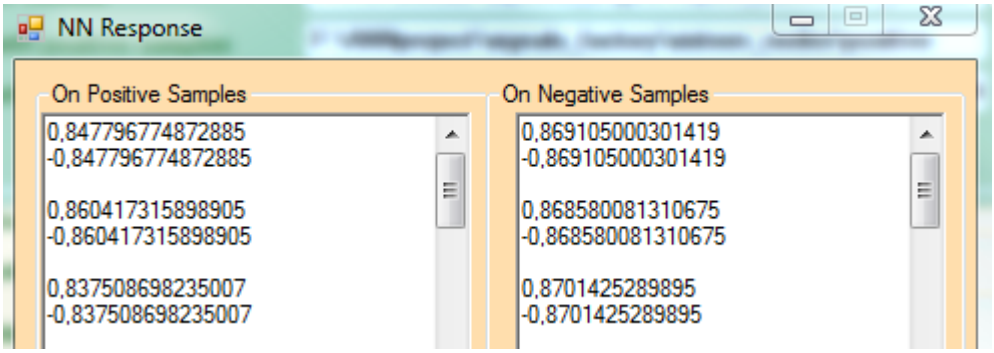


Рис.3.46. Однакова реакція на P і N2 як на позитивні сигнали

Тепер подамо в якості позитивного сигналу неполадку N1, а в якості негативного сигналу - неполадку N2. В цьому випадку мережа очікувано не може розпізнати жоден із сигналів (рис.3.47).

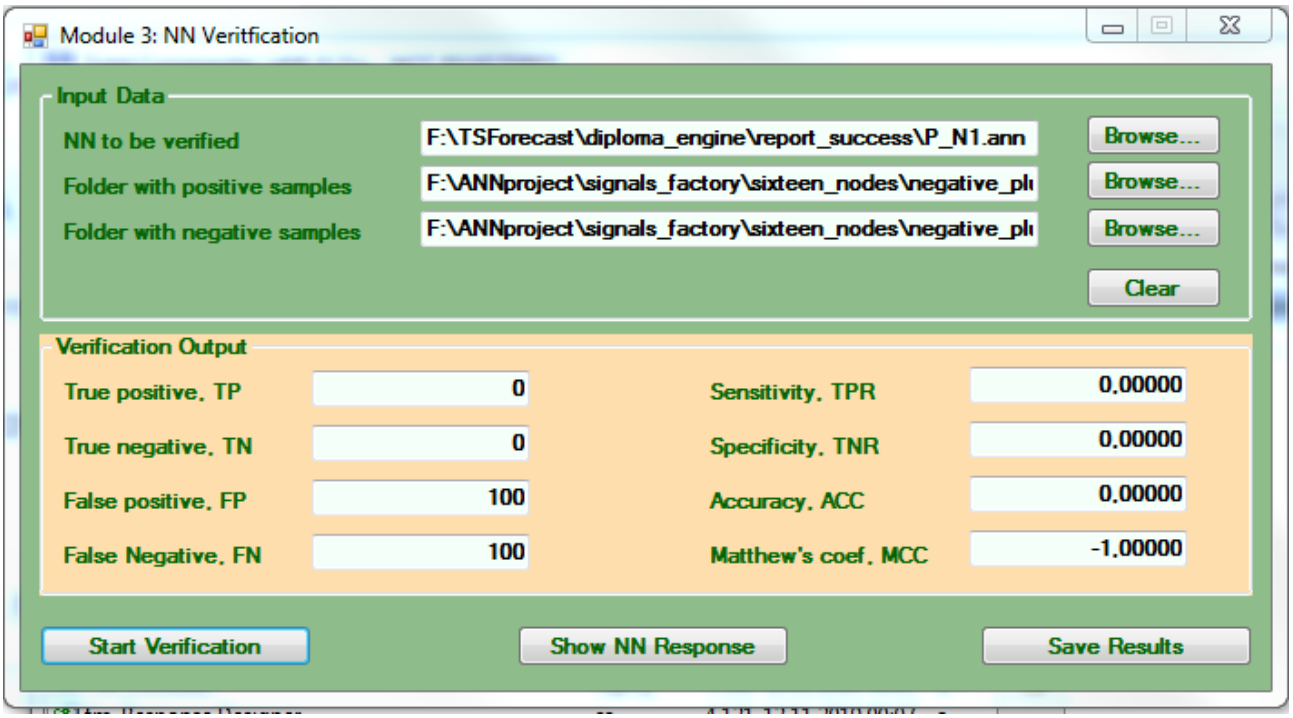


Рис.3.47. Нейромережа, навчена на сигналах P і N1,
не може розпізнати жоден із сигналів із пари N1 і N2

Як видно із сигналів на виході (рис.3.48), вхідні сигнали N1 в цьому випадку будуть розпізнані як False Negative, а сигнали N2 – як False Positive.

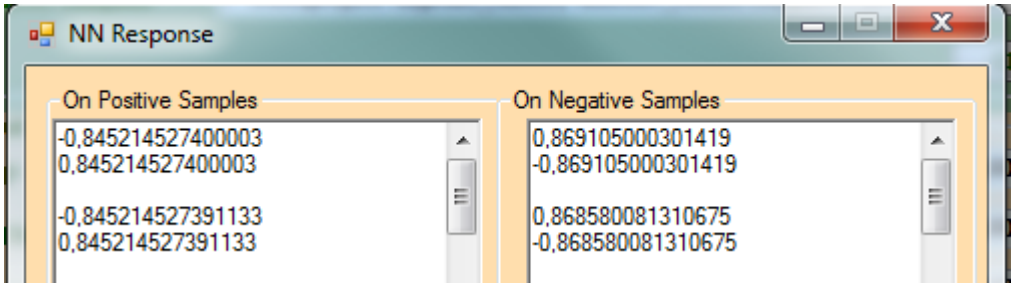


Рис.3.48. Реакція на N1 і N2

Таким чином, запропонована методика верифікації показує релевантні результати і може бути використана для проведення досліджень і підтвердження якості навчених нейромереж.

Використовуючи Модуль 3 ми відібрали із 45 навчених на Етапі 2 нейромереж (по 3 на кожну пару сигналів) 15 кращих нейромереж для кінцевого етапу дослідження – побудови каскаду.

3.4. Розпізнавання множинних сигналів за допомогою каскаду із навчених нейронних мереж

Вхідними даними для метаструктури (каскаду), що розпізнає всю множину вхідних сигналів, є навчені на парах сигналів нейромережі (див. табл.1.6). Для побудови мета структури та роботи з нею призначено Модуль 4 пакету TSForecast (рис.3.49).

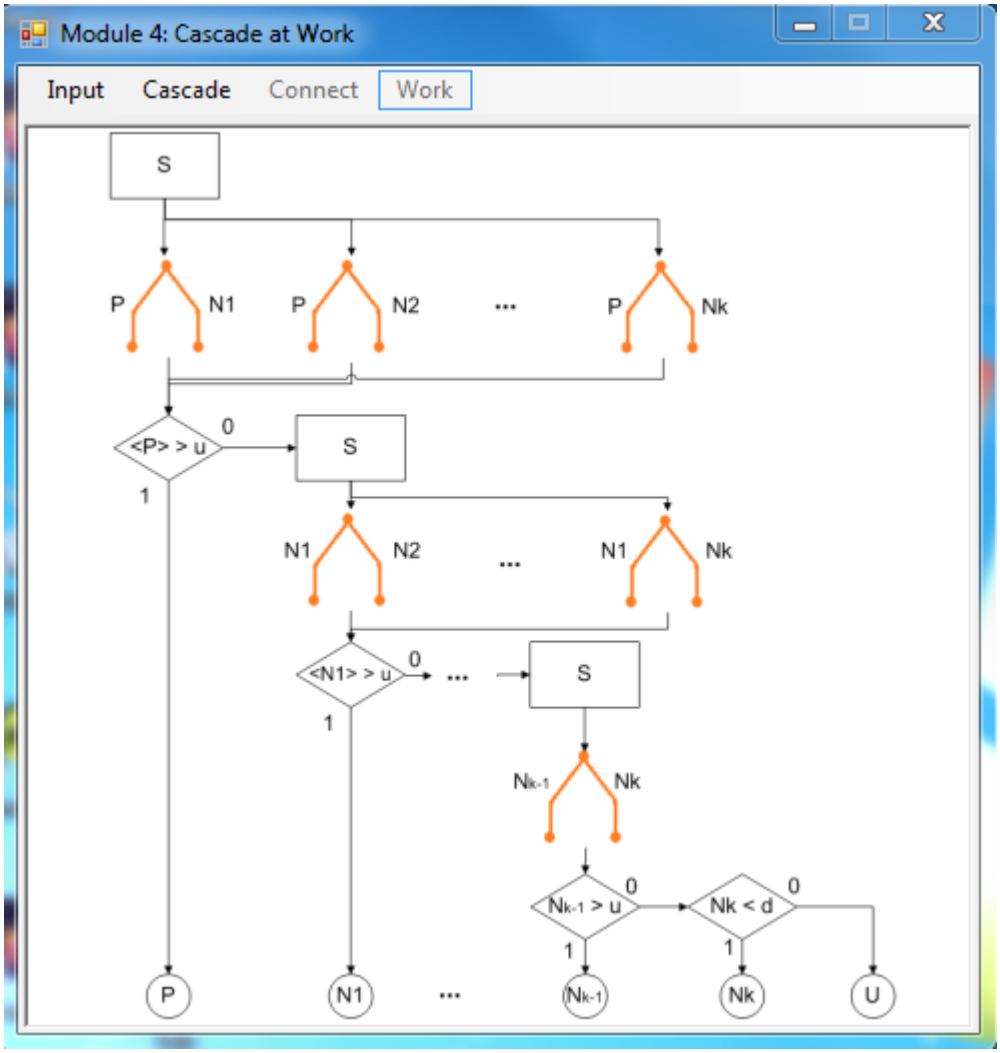


Рис.3.49. Головна форма Модуля 4 пакету TSForecast

Головне меню має 4 розділи: Input, Cascade, Connect, Work.

Після вибору в головному меню опції Input з’являється діалогове вікно із запитом про число неполадок, які будуть досліджуватися (рис.3.50).

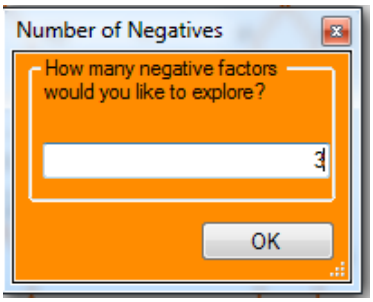


Рис.3.50. вікно із запитом про кількість неполадок, що досліджуються

Нехай досліджуються 3 неполадки. Вводимо їх число, натискаємо ОК і отримуємо форму для введення навчених нейромереж (по парам сигналів) та відповідних папок із зразками, що будуть представлені каскаду (рис.3.51).

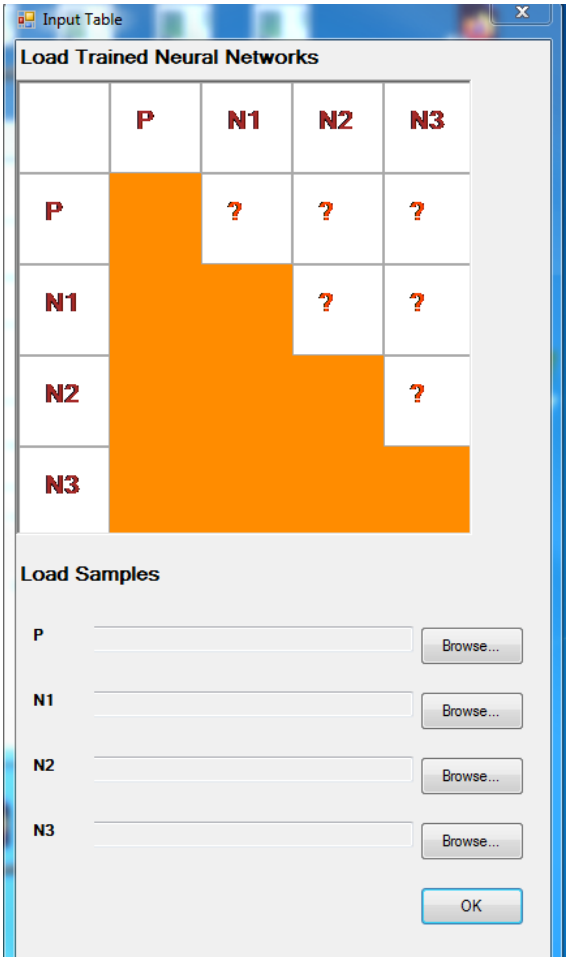


Рис.3.51. Форма для введення вихідних даних для побудови каскаду

Робимо подвійний клік на першому знаку питання «?» і в системному файловому діалозі розміщаємо навчену на парі «P-N1» нейромережу (рис.3.52).

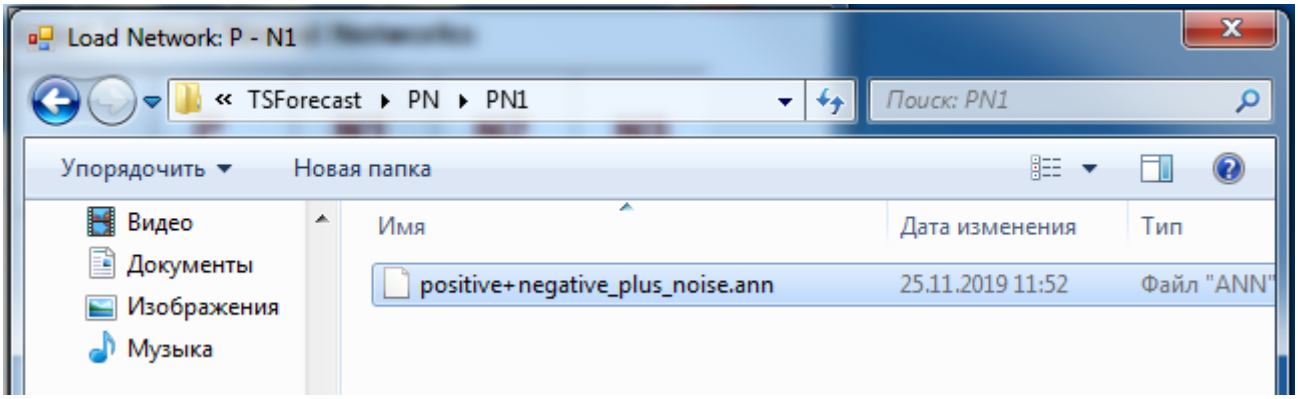


Рис.3.52. Завантаження нейромережі навченої на парі «P-N1»

У формі замість знаку питання виникає відмітка «√» (рис.3.53).

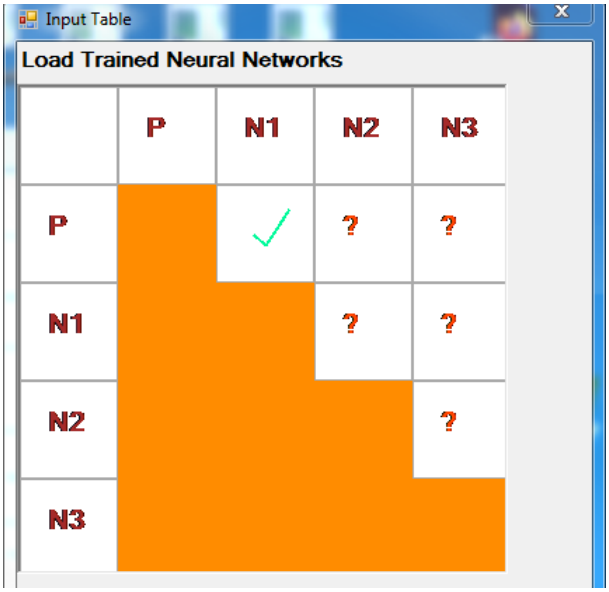


Рис.3.53. Нейромережу завантажено

Аналогічно завантажуємо решту нейромереж.

Далі – завантажуємо папки із зразками, що відповідають досліджуваним сигналам (не парним, а окремим). Після заповнення всієї форми (рис.3.54), натискаємо кнопку ОК.

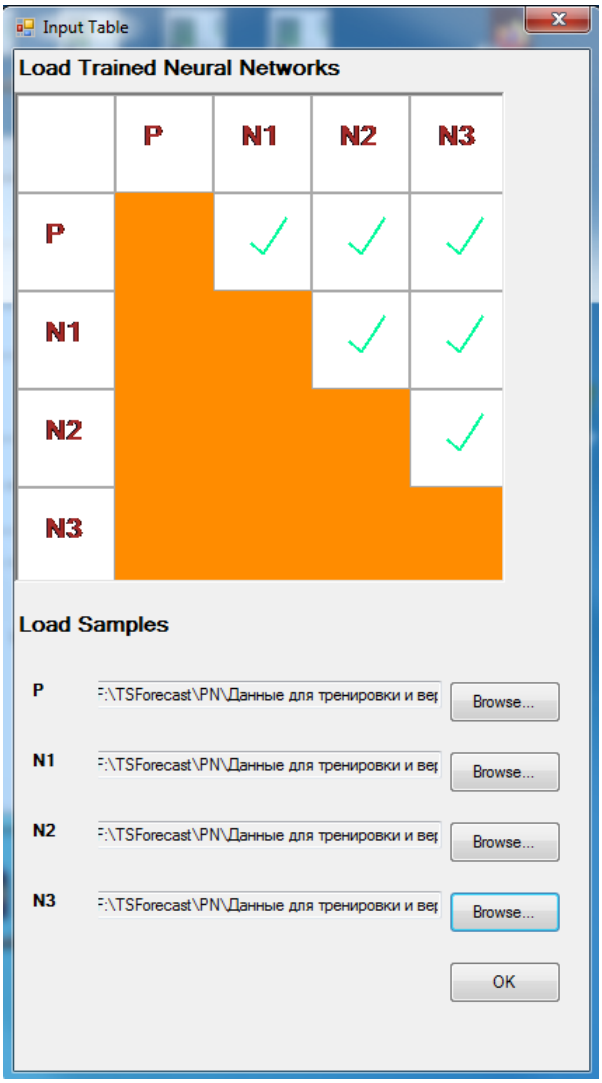


Рис.3.54. Заповнена форма із вхідними даними

Отримуємо повідомлення про створення каскаду та рекомендацію щодо його зберігання (рис. 3.55).

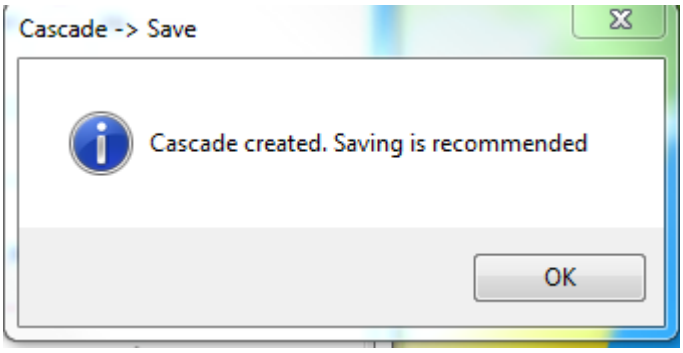


Рис.3.55. Повідомлення про створення каскаду

У розділі Cascade доступні дві опції: Load та Save.

Обираємо Cascade → Save та зберігаємо каскад у бінарному файлі (рис.3.56).

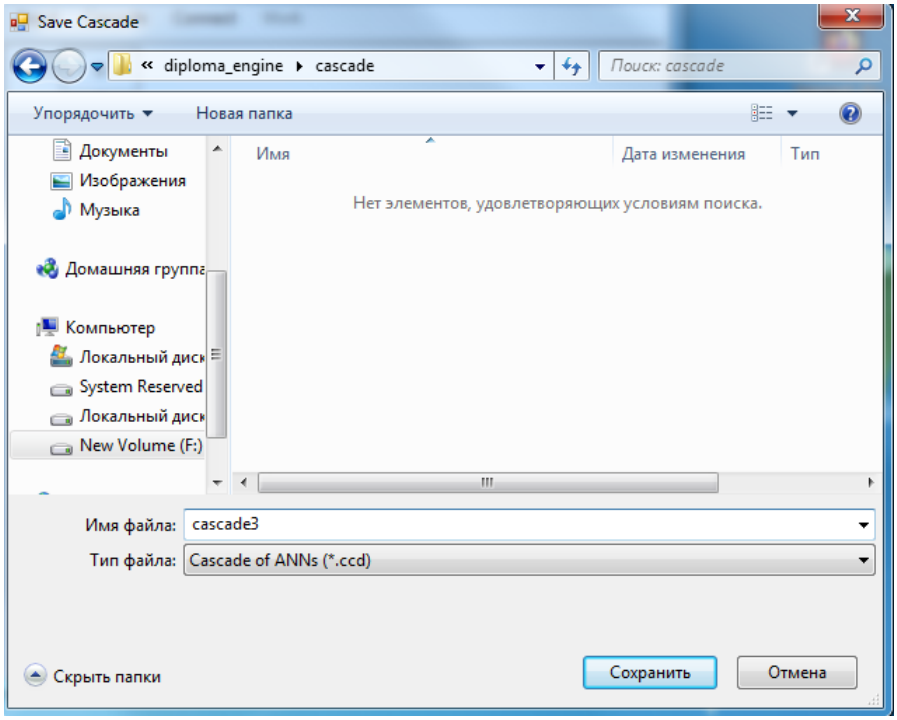


Рис.3.56. Зберігання каскаду у бінарному файлі

Після того, як каскад створено (або завантажено із файлу), стає доступною опція Connect. Натискаємо на неї та отримуємо повідомлення про підключення всіх нейромереж до каскаду та готовність до роботи (рис.3.57).

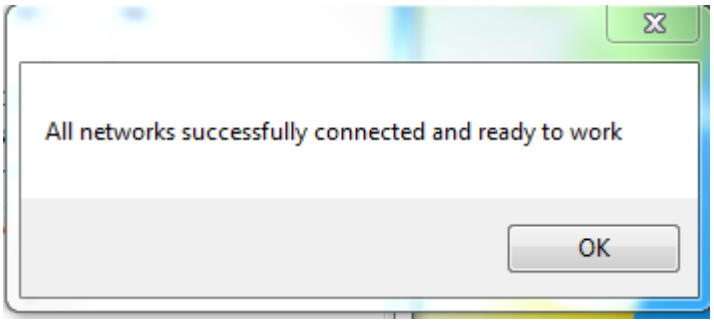


Рис.3.57. Повідомлення про підключення нейромереж до каскаду

Підключення полягає в приєднанні об’єктів типу NeuralNet, що зберігаються у файлах, шляхи до котрих збережені у об’єкті типу Cascade.

Після цього стає доступною опція Work.

Натискаємо на неї і отримуємо вікно із запитом про режим роботи: з презентацією одного сигналу (вибирається випадково із підключених зразків) чи з статистикою по всім сигналам (рис.3.58).

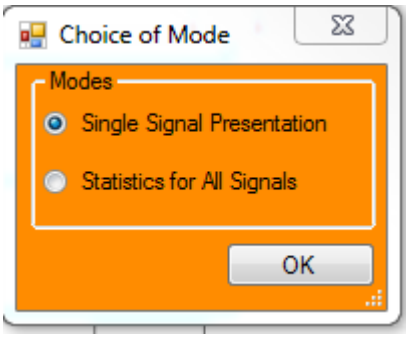


Рис.3.58. Вибір режиму роботи

Обираємо режим презентації, отримуємо звіт у формі зображення (рис.3.59).

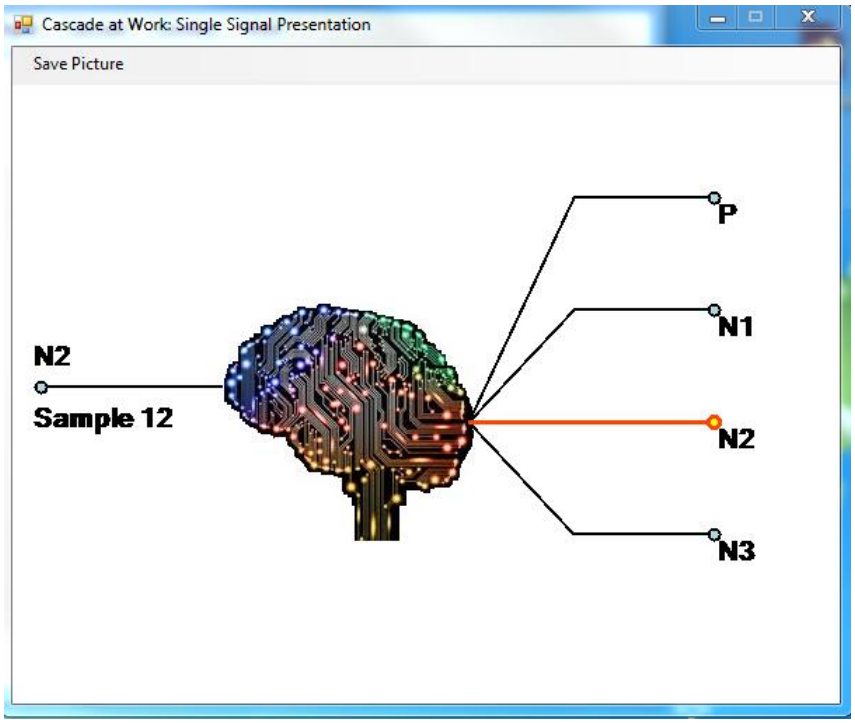


Рис.3.59. Розпізнавання випадкового сигналу у режимі презентації (позитивний сигнал та три неполадки)

Згідно отриманого зображення, на вхід було подано зразок 12, що відповідає сигналу N2 (надписи зліва). Після розпізнавання, каскад («мозок») визначив сигнал як N2 (надпис справа). Отже, розпізнавання відбулося вірно.

Обравши опцію Save Picture, зображення можна зберегти в одному із трьох форматів (bmp, jpg, png).

Тепер оберемо режим збору статистики (подача на входи каскаду всієї множини зразків). Отримуємо повідомлення, що статистику зібрано (рис.3.60).

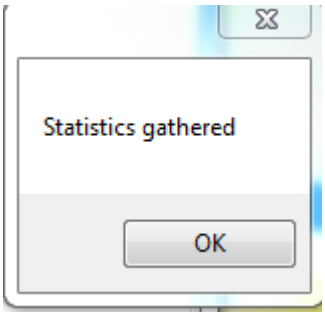


Рис.3.60. Повідомлення про успішний збір статистики

Натискаємо ОК і отримуємо статистику розпізнавання штатної роботи та трьох неполадок за допомогою каскаду (рис.3.61).

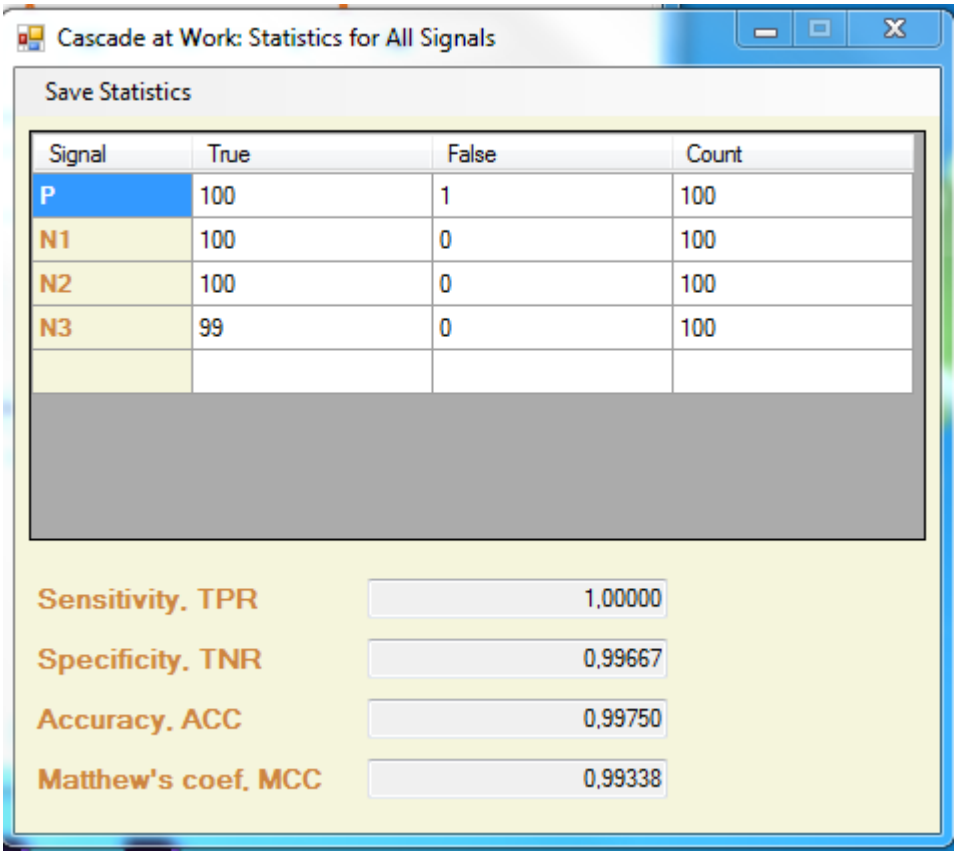


Рис.3.61. Результат розпізнавання штатної роботи
та трьох неполадок за допомогою каскаду

Як бачимо, один із сигналів N3 був помилково визначений як P. Всі інші сигнали визначені вірно.

Загальна точність отриманого каскаду ACC = 0,9975 (99,75%).

Таким чином, запропонована методика побудови метаструктури (каскаду) над нейромережами, котрі були навчені розпізнавати пари сигналів, дозволяє отримати систему, знатну до множинного розпізнавання з великою точністю.

Закінчимо дослідження побудовою та використанням каскаду, здатного розпізнавати всі наявні сигнали – штатну роботу та 5 неполадок.

Кроки побудови каскаду на 5 неполадок аналогічні крокам розглянутим вище (див. рис.3.49-3.57). Після того як каскад створено та збережено у файлі, проведемо його опробування.

Робота у режимі презентації представлена на рис.3.62.

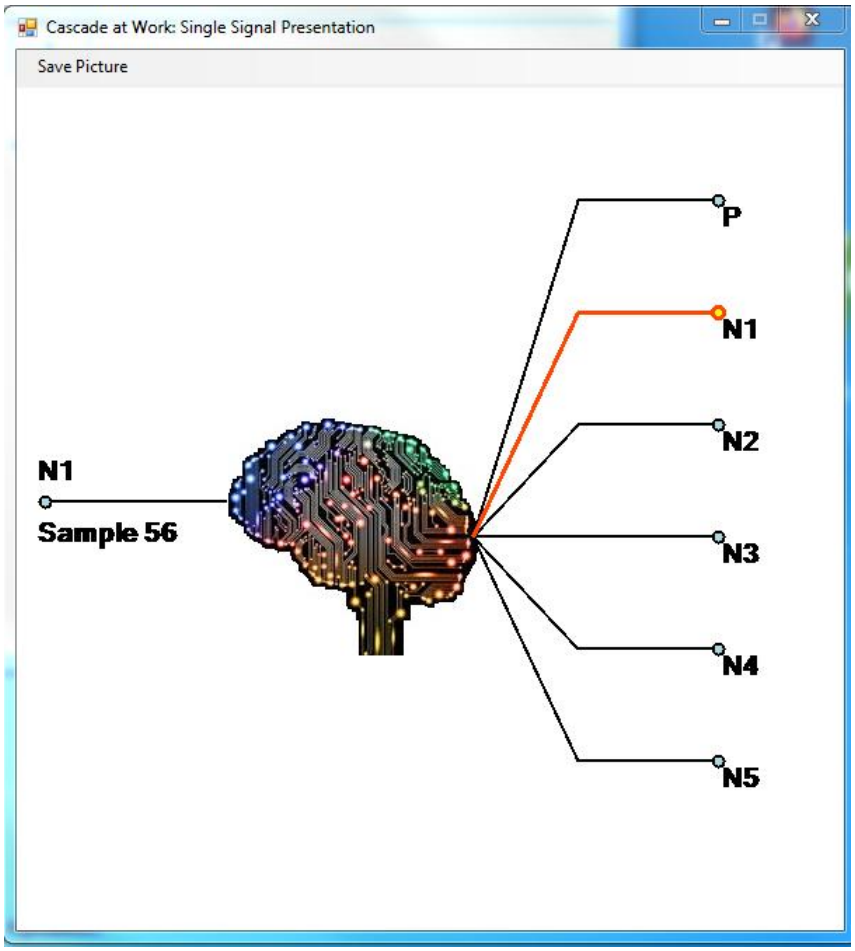


Рис.3.62. Розпізнавання довільного сигналу у режимі презентації (позитивний сигнал та п’ять неполадок)

На вхід подано довільно обраний сигнал (зразок 56, неполадка N1), котрий успішно розпізнається.

Зберемо статистику для каскаду на 5 неполадок на всій множині зразків (рис.3.63).

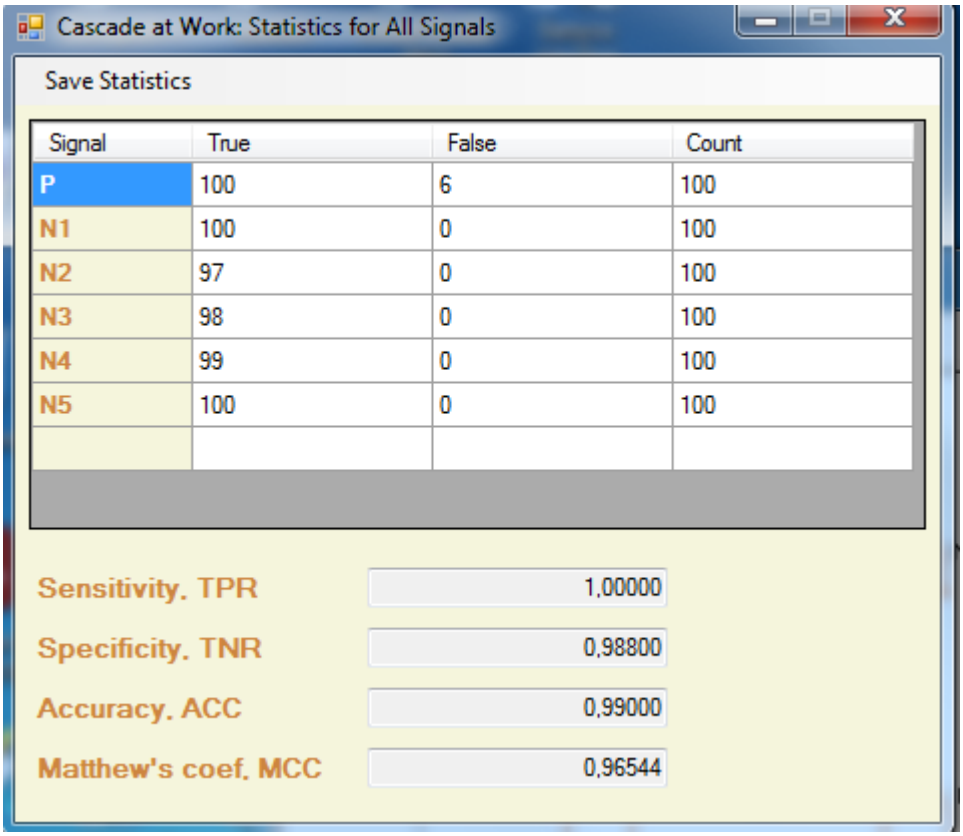


Рис.3.63. Результат розпізнавання штатної роботи та п’яти неполадок за допомогою каскаду нейромереж

Побудований із 15 нейромереж каскад, що здатний розпізнавати штатну роботу та 5 неполадок, показав наступні результати: чутливість 1,000; селективність 0,988; точність 0,990; коефіцієнт Метьюза 0,965. Таким чином, отримана система демонструє дуже високу якість розпізнавання, що вказує на перспективність запропонованого напрямку досліджень.

ВИСНОВКИ

На основі проведеного дослідження можна зробити наступні висновки.

У даній роботі пропонується система на нейронних мережах, що здатна розрізняти штатну роботу та декілька аварійних ситуацій із високою точністю. Сигнали для аналізу поступають від датчика частоти обертів (тахогенератора), встановленого на працюючому двигуні. Сигнал від тахогенератора поступає на комп'ютер, обробляється і записується у файли даних (значення напруги) за установлений час.

Отриманий сигнал перед подачею на вхід нейронної мережі повинен згортатися. Концепція згортки запропонована в даному дослідженні, побудована на моделюванні сигналу як композиції «гармоніка + білий шум», тобто квазігармонійного сигналу. Вважатимемо сигнал *позитивним* (нормальним), якщо відношення шуму до сигналу не перевищує 5%. *Неполадка* (негативний сигнал) – це систематичне відхилення сигналу системи від норми. В роботі вводиться 5 неполадок: 1) зростання рівня шуму, 2) зростання амплітуди, 3) зниження амплітуди, 4) зростання частоти, 5) зниження частоти.

Для будь-якого квазігармонічного процесу, який описується дискретним сигналом довільної тривалості, ми пропонуємо формувати характеристичний вектор із 16 складових, які в сукупності описують амплітуду, частоту, потужність основного сигналу, характеристики шуму та показники варіації.

Переваги даного підходу перетворення сигналу для класифікації за допомогою нейромереж:

1) врахування фізичної природи сигналу;

2) незалежність від тривалості сигналу. В одній вибірці можуть знаходитися сигнали різної тривалості, але тривалість не впливає на зміст компонент 16-вектора, що застосовуються для навчання нейромереж;

3) незалежність від фази сигналу, що надійшов. В одній вибірці можуть бути квазігармонічні сигнали з різними фазами, але фаза не впливає на зміст компонент 16-вектора;

4) значне скорочення розмірності вхідних даних (згортка багатокомпонентного вектора довільної розмірності в 16-компонентний вектор).

Характеристичні 16-вектори, що подаються на вхід нейронних мереж для навчання, мають бути нормовані. Завдяки особливостям функцій активації нейронів, вибір еталонного зразка та нормування на нього є обов'язковим етапом підготовки зразків до навчання нейронних мереж.

Розмірність характеристичних 16-векторів визначає розмірність вхідного S-шару нейронних мереж, котра має бути кратною 16. Характеристичні вектори можна подавати послідовно на нейрони входу один або декілька разів.

Кількість прихованих шарів (А-шари) і число нейронів в них є довільним. В процесі експериментів можна знайти найбільш доцільну топологію з точки зору компромісу між швидкістю та якістю навчання. Складна топологія знижує швидкість, але в цілому підвищує якість, і навпаки.

З метою досягнення високої точності та надійності, необхідної для систем реального часу, кожна нейронна мережа проектується як класифікатор двох подій; наприклад, має розрізняти штатну роботу від неполадки N1, або неполадку N2 від неполадки N5 тощо. Відповідно, на виході нейронної мережі (R-шар) повинно бути 2 нейрони.

При навчанні нейромережі у стохастичному режимі всі доступні зразки подаються на у рандомному порядку протягом епохи. *Епоха* – це одна ітерація у процесі навчання, під час якої нейронній мережі пред'являються всі зразки із навчальної множини. Кількість епох задається користувачем. Для управління процесом зворотного розповсюдження помилки використовуються два параметри: швидкість навчання η і момент α.

Візуально, успіх або невдача при навчанні можуть бути оцінені за графіком залежності цільової функції $E(n)$ від номеру епохи. При успішному навчанні значення $E(n)$ прямує до 0. Після вдалого за візуальною оцінкою навчання кожна нейронна мережа повинна проходити процедуру верифікації. Для цього використовують частину отриманих зразків (30-50%), котрих нейронна мережа «не бачила» під час навчання. Якісно навчена мережа має класифікувати ці – нові для себе – сигнали із заданою точністю.

Процедура верифікації побудована на статистиці розпізнавань із показниками TP, TN, FP, FN. За ними розраховуються показники чутливості, селективності, точності та коефіцієнт Метьюза.

Для кожної пари подій із множини штатної роботи та неполадок відбираються найбільш вдало навчені нейронні мережі, на котрих будується метаструктура, яка за комбінаторним принципом здатна розрізняти всі події, визначені у множині.

В роботі запропоноване об’єднання навчених нейромереж (класифікаторів пар сигналів) у каскад – «надмережу», здатну розпізнавати всю множину досліджуваних сигналів. При подачі сигналу на вхід каскаду опитуються всі нейромережі. При цьому ті із них, що були навчені розпізнавати саме цей сигнал, дають більший за абсолютним значенням відгук. В результаті підрахунку суми «голосів» сигнали розпізнаються з високою точністю.

Перевагами побудови каскаду над нейромережами, що є бінарними класифікаторами, є

1) висока ступінь контролю над навчанням окремих компонент – нейромереж;

2) посилення вихідного сигналу за рахунок участі тих нейромереж, що навчені розпізнавати сигнал на вході;

3) можливість верифікації самого каскаду за статистичними показниками;

4) висока якість множинного розпізнавання при умові якісних окремих компонент – нейромереж.

У практичній частині дослідження нами було розроблено пакет TSForecast для автоматизації роботи на всіх етапах дослідження. В пакет входять 4 модулі, розроблені на мові C#, та скрипти на MATLAB для підготовчого етапу.

На підготовчому етапі – нарізка файлів із сигналами на зразки та згортка у 16-сектори застосовуються скрипти MATLAB. Було отримано 600 згорток – по 100 на кожний із видів сигналів. За рахунок використання паралельних розрахунків та оптимізації MATLAB при роботі з масивами та файловою системою, всі операції виконуються дуже швидко (1-2 секунди), незважаючи на значні обсяги даних (до 10^6 точок).

Для візуалізації вхідних даних, їх згладжування та нормалізації призначено додаток Модуль 1 пакету TSForecast. В модулі можна провести лінійне та експоненційне згладжування, лінійну та нелінійну нормалізацію у відрізки [0;1] або [-1;1].

Для створення нейромереж, їх навчання та попереднього оцінювання якості призначено Модуль 2. Користувач може задати наступні параметри архітектури: кількість внутрішніх A-шарів, кількість нейронів у S-шарі та в кожному із A-шарів, функцію активації для нейронів. Для навчання можна обрати режим (стохастичний, упорядкований або змішаний), задати кількість епох, швидкість навчання η та момент навчання α. Після навчання можна отримати звіт, що складається із графіку залежності цільової функції (середньоквадратичного відхилення) від номеру епохи та текстової частини, де сказані всі визначені користувачем параметри,початкове і кінцеве значення СКВ. Нейронну мережу зберігають у бінарному файлі, звіт – у текстовому файлі. За допомогою Модуля 2 було навчено і відібрано за попереднім оцінюванням 45 нейромереж (по 3 на кожну із 15 пар сигналів).

Для процесу верифікації у пакеті TSForecast розроблено Модуль 3. Користувач обирає нейромережу та призначає папки з позитивними та негативними зразками, на яких буде проходити верифікація. Для цього із файлів з первинними даними нарізаються нові зразки та отримуються згортки із

16-векторів (знову 600 зразків – по 100 зразків на кожний вид сигналу). За результатами розпізнавання збирається статистика та розраховуються статистичні показники. Також, у цьому модулі можна побачити вихідні сигнали на обох нейронах R-шару, що є корисним для оцінювання якості навченої нейромережі. За результатами верифікації було остаточно відібрано 15 нейромереж для кожної пари досліджуваних сигналів.

Для побудови метаструктури (каскаду) та роботи з нею призначено Модуль 4 пакету TSForecast. При збиранні навчених нейромереж у каскад перед користувачем постає досить важка задача: підключити нейромережі за комбінаторним принципом та приєднати відповідні зразки для верифікації (це може бути десятки різних нейромереж та десятки папок, в кожній по сотням зразків). Дизайн головної форми Модуля 4 допомагає швидко і якісно вирішити цю задачу.

Результати роботи каскаду можна подивитися в режимі презентації, де на вхід подається довільно обраний сигнал і візуально демонструється як він розпізнається. Для практичних цілей переходимо у режим збору статистики, де отримуємо оцінку якості всієї розробленої системи.

Побудований із 15 нейромереж каскад, що здатний розпізнавати штатну роботу та 5 неполадок, показав наступні результати: чутливість 1,000; селективність 0,988; точність 0,990; коефіцієнт Метьюза 0,965. Таким чином, отримана система демонструє дуже високу якість розпізнавання, що вказує на перспективність запропонованого напрямку досліджень.

На нашу думку, точність порядку 95-99% і надійність пропонованого множинного каскадного класифікатора має зацікавити фахівців-практиків, що планують впровадження систем штучного інтелекту на виробництві.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Alba E., Marti R. (Eds.) Metaheuristic Procedures for Training Neural Networks. - Springer, 2006. — 258 pp. — ISBN: 978-0387-33415-8.
2. Aldrich C. Exploratory Analysis of Metallurgical Process Data with Neural Networks and Related Methods. - Elsevier, 2002. — 387 p.
3. Anastassiou G.A. Intelligent Systems II. Complete Approximation by Neural Network Operators. - Springer, 2016. — 712 p.
4. Anthony M., Bartlett P.L. Neural Network Learning: Theoretical Foundations. - Cambridge University Press, 2009. - 389 p.
5. Buda, M., Maki, A., and Mazurowski, M. A. (2018). A systematic study of the class imbalance problem in convolutional neural networks. Neural Netw. 106, 249–259. doi: 10.1016/j.neunet.2018.07.011
6. George, G.A.; Landryova, L. Interaction between human and AI systems—When automated systems move towards autonomous. In Proceedings of the 2019 20th International Carpathian Control Conference, ICCC 2019, Wieliczka, Poland, 26–29 May 2019.
7. Ghahramani, M.; Qiao, Y.; Zhou, M.; Hagan, A.O.; Sweeney, J. AI-based modeling and data-driven evaluation for smart manufacturing processes. IEEE/CAA J. Autom. Sin. 2020, 7, 1026–1037.
8. Khan, A., Sohail, A., Zahoora, U., and Qureshi, A. S. (2019). A survey of the recent architectures of deep convolutional neural networks. arXiv:1901.06032.
9. Kebria, P.M.; Khosravi, A.; Salaken, S.M.; Nahavandi, S. Deep imitation learning for autonomous vehicles based on convolutional neural networks. IEEE/CAA J. Autom. Sin. 2020, 7, 82–95.
10. Landryova, L., Sikora J., Wagnerová R. The Learning Path to Neural Network Industrial Application in Distributed Environments // Processes 2021, 9, 2247. https://doi.org/10.3390/pr9122247
11. Li, Q.; Cai, W.; Wang, X.; Zhou, Y.; Feng, D.D.; Chen, M. Medical image classification with convolutional neural network. In Proceedings of the 2014

13th International Conference on Control Automation Robotics & Vision (ICARCV), Singapore, 10–12 December 2014; pp. 844–848.

12. Ma, Y.;Wang, Z.; Yang, H.; Yang, L. Artificial intelligence applications in the development of autonomous vehicles: A survey. IEEE/CAA J. Autom. Sin. 2020, 7, 315–329
13. Miskuf, M.; Zolotova, I. Application of business intelligence solutions on manufacturing data. In Proceedings of the 2015 IEEE 13th International Symposium on Applied Machine Intelligence and Informatics (SAMI), Herlany, Slovakia, 22–24 January 2015; pp. 193–197.
14. Patan K., Korbicz J. Application of Dynamic Neural Networks in an Industrial Plant // 4th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes 2000 (SAFEPROCESS 2000), Budapest, Hungary, 14-16 June 2000 // https://doi.org/10.1016/S1474-6670(17)37358-5
15. Radosavovic Ilija, Kosaraju Raj Prateek, Girshick Ross, He Kaiming, Dollár Piotr. Designing Network Design Spaces // https://arxiv.org/pdf/2003.13678.pdf
16. Ramchoun, H.; Idrissi, M.A.J.; Ghanou, Y.; Ettaouil, M. Multilayer Perceptron: Architecture Optimization and Training. Int. J. Interact. Multimed. Artif. Intell. 2016, 4, 26.
17. Smith, L. N. (2018). A disciplined approach to neural network hyper-parameters: Part 1 – learning rate, batch size, momentum, and weight decay. arxiv:1803.09820.
18. Tan Mingxing, Le Quoc V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks // https://arxiv.org/abs/1905.11946
19. Young, T.; Hazarika, D.; Poria, S.; Cambria, E. Recent Trends in Deep Learning Based Natural Language Processing. IEEE Comput. Intell. Mag. 2018, 13, 55–75.

ДОДАТКИ

Додаток А

Зведена таблиця для нормування характеристичних 16-векторів

№	Розрахункова формула	Нормування
1	$x_{max} = \max_i x(t_i)$	$x_{max}/x_{\max et}$
2	$x_{min} = \min_i x(t_i)$	$x_{min}/x_{\min et}$
3	$\langle x \rangle = \frac{1}{N}\sum_{i=1}^{N} x(t_i)$	$\langle x \rangle/\langle x \rangle_{et}$
4	$v_{max} = \max_i v(t_i)$	$v_{max}/v_{\max et}$
5	$v_{min} = \min_i v(t_i)$	$v_{min}/v_{\min et}$
6	$\langle v \rangle = \frac{1}{N-1}\sum_{i=2}^{N} v(t_i)$	$\langle v \rangle/\langle v \rangle_{et}$
7	$a_{max} = \max_i a(t_i)$	$a_{max}/a_{\max et}$
8	$a_{min} = \min_i a(t_i)$	$a_{min}/a_{\min et}$
9	$\langle a \rangle = \frac{1}{N-2}\sum_{i=3}^{N} a(t_i)$	$\langle a \rangle/\langle a \rangle_{et}$
10	$\langle \omega \rangle = \frac{v_{max} - v_{min}}{x_{max} - x_{min}}$	$\langle \omega \rangle/\langle \omega \rangle_{et}$
11	$\langle \omega^2 \rangle = \frac{a_{max} - a_{min}}{x_{max} - x_{min}}$	$\langle \omega^2 \rangle/\langle \omega^2 \rangle_{et}$
12	$\langle P \rangle = \frac{(x_{max} - x_{min})^2}{8}$	$\langle P \rangle/\langle P \rangle_{et}$
13	$\sigma_x = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x(t_i) - \langle x \rangle)^2}$	$\sigma_x/\sigma_{x\,et}$
14	$\sigma_v = \sqrt{\frac{1}{N-2}\sum_{i=2}^{N}(v(t_i) - \langle v \rangle)^2}$	$\sigma_v/\sigma_{v\,et}$
15	$\sigma_a = \sqrt{\frac{1}{N-3}\sum_{i=3}^{N}(a(t_i) - \langle a \rangle)^2}$	$\sigma_a/\sigma_{a\,et}$
16	$d_p = \left\lvert \frac{1}{2N}\sum_{i=1}^{N} x^2(t_i) - \langle P \rangle \right\rvert$	$d_p/d_{p\,et}$

Додаток Б

Скрипт на мові MATLAB для нарізки файлу з даними

```
%cutting a datafile into fragments

%-------------------------- initial data
T = 1000;    %length of each fragment
N = 150;     %number of fragments
h = 50;      %shift rightwards for next fragment

%---------------------- open file dialog
[file, path] = uigetfile('*.txt');
if isequal(file,0)
    f1 = msgbox('No datafile chosen','File Dialog','error');
else
    full_name = fullfile(path,file);
    datafileID = fopen(full_name,'r');
    X = fscanf(datafileID,'%f');
    t = size(X,1);

    %--------------------check the possibility
    if(h*N + T <= t)
        %-------- folder dialog
        folder_name = uigetdir('','Choose/Create Folder for Fragments');
        if isequal(folder_name,0)
            f2 = msgbox('No folder chosen','Folder Dialog','error');
        else
            file_prefix = 'frag';

            for k = 1:N
                %cut fragment
                Y = X(h*k:h*k+T);
                %save fragment
                frag = [file_prefix num2str(k,'%03d') '.txt'];
                f_name = fullfile(folder_name,frag);
                f_ID = fopen(f_name,'w');
                fprintf(f_ID,'%.10f\r\n',Y);
                fclose(f_ID);
            end
        end

        h = msgbox('All fragments successfully created');
    else
        f = msgbox('Datafile is too short for the specified number of
fragments');
    end
end
```

Додаток В

Множинна згортка: папка із зразками сигналів у папку із 16-векторами

```
%folder of samples to folder of 16-vectors

%------- open folder with samples
folder_samples = uigetdir('','Choose Folder with Samples');
if isequal(folder_samples,0)
    f1 = msgbox('No folder chosen','Folder Dialog','error');
else
    %------- specify folder for vectors
    folder_vectors = uigetdir('','Specify Folder for Vectors');
    if isequal(folder_vectors,0)
        f1 = msgbox('No folder specified','Folder Dialog','error');
    else
        sample_files = dir(fullfile(folder_samples,'*.txt'));
        N = size(sample_files, 1);

        %if empty
        if isequal(N, 0)
            n1 = msgbox('The chosen folder is empty');
        else
            %-----------------choose etalon file
            [etalon_file, path] = uigetfile('*.txt');
            if isequal(etalon_file,0)
                f2 = msgbox('No etalon chosen','File Dialog','error');
            else
                full_etalon = fullfile(path,etalon_file);
                etalonID = fopen(full_etalon,'r');
                X0 = fscanf(etalonID,'%f');
                V0 = Vector16(X0);

                %------------------process all samples
                samples_folder = sample_files(1).folder;
                file_prefix = 'vec';
                for k = 1:N
                    current_sample = fullfile(samples_folder,
sample_files(k).name);
                    sampleID = fopen(current_sample,'r');
                    X = fscanf(sampleID,'%f');
                    V = Vector16(X);
                    %normalization
                    for i = 1:16
                        V(i) = V(i)/V0(i);
                    end

                    %save normalized vector
                    vector = [file_prefix num2str(k,'%03d') '.txt'];
                    f_name = fullfile(folder_vectors,vector);
                    f_ID = fopen(f_name,'w');
                    fprintf(f_ID,'%.10f\r\n',V);
                    fclose(f_ID);
                end
            end
            h = msgbox('All vectors successfully created');
        end
    end
end
```