Міністерство освіти і науки України
Криворізький національний університет
Факультет інформаційних технологій
Кафедра комп’ютерних систем та мереж

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА
за спеціальністю 123 - «Комп’ютерна інженерія»

на тему:

МЕТОДИ НАВЧАННЯ ОБЧИСЛЮВАЛЬНИХ АРХІТЕКТУР НА ОСНОВІ ШТУЧНИХ НЕЙРОПРОЦЕСОРІВ

Проектував _____ Д. А. Лебідь

Керівник випускної роботи _____ А. І. Купін

Нормоконтроль _____ Д. І. Кузнєцов

Завідувач кафедри _____ А. І. Купін

Кривий Ріг
2022

Ступінь вищої освіти магістр
Спеціальність 123 «Комп’ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри, голова циклової комісії

_____ А. І. Купін

“____” _____ 20__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім’я, по батькові)

1. Тема роботи _____
_____
_____
керівник роботи _____,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом вищого навчального закладу від “___”_____ 20__року №__
2. Строк подання студентом роботи _____
3. Вихідні дані до роботи _____
_____
_____
_____
_____
_____
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)_____
_____
_____
_____
_____
_____
5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)_____
_____
_____
_____
_____

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

РЕФЕРАТ

Змн.	Арк.	№ документа	Підпис	Дата			
Розробив		Лебідь			РЕФЕРАТ	Літера	Аркуш	Аркушів
Перевірив		Купін						
						КІ-21М		
Н.контроль		Кузнєцов						
Затвердив		Купін						

Пояснювальна записка: 58 с., 23 рисунки, 2 табл., 1 додаток, 22 літературних джерела.

Об'єкт розробки – архітектура обчислювальної системи на основі нейронної мережі радіально-базисних функцій.

Мета роботи: розробити архітектуру обчислювальної системи на основі нейронної мережі радіально-базисних функцій типу нейрокомп’ютер.

В загальній частині зроблений огляд теорії нейромереж та апаратного забезпечення сучасних обчислювальних систем на основі теорії нейромереж.

В проектній частині виконано вибір апаратної бази та створено схеми обчислювальної системи, а також виконано аналіз архітектури систем типу нейронні мережі радіально-базисних функцій.

Результатом роботи архітектура нейрокомп’ютера для роботи нейронної мережі радіально-базисних функцій.

НЕЙРОМЕРЕЖА, НЕЙРОКОМП’ЮТЕР, КЛАСТЕР, ШТУЧНИЙ ІНТЕЛЕКТ, ОБЧИСЛЮВАЛЬНА СИСТЕМА.

Explanatory note: 58 pp., 23 illustrations, 2 tab., 1 appendix, 22 literary sources.

Facility design - architecture computing systems based on neural network of radial basis functions.

Objective: To develop the architecture of computer systems based on neural network of radial basis functions such neurocomputer.

In general overview of the theory of neural networks and hardware of modern computing systems based on neural network theory.

In the design of the choice made hardware base and established circuits of computer systems and the analysis of the architecture of such neural networks radial basis functions.

The result of the neural computer architecture for neural networks radial basis functions.

NEURAL NETWORKS, NEURAL COMPUTERS, CLUSTERS, ARTIFICIAL INTELLIGENCE, COMPUTER SYSTEM.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

НК – нейронний комп’ютер;
НО - нейронний обчислювач;
ОСМП - обчислювальні системи з масовим паралелізмом;
ПЛІС – програмуємо-логічні інтегральні схеми;
ППО – паралельний перепрограмуємий обчислювач;
AI - Artifical Intelligence;
NM – NeuroMatrix;
BP - Back Propagation;
ЕОМ – електронно-обчислювальна машина;
MPI - Message Passing Interface.

ВСТУП

Темою випускної роботи є «Архітектура обчислювальної системи на основі нейронної мережі радіально-базисних функцій».

Актуальність роботи.

На сьогоднішній день штучний інтелект залишається одним із найбільш перспективних і нерозкритих напрямків розвитку інформаційних управляючих систем та технологій. До складу понять штучного інтелекту сьогодні відносять нейронні мережі, нечітку логіку, експертні системи, ЕОМ п'ятого покоління, системи моделювання мислення.

Мета і завдання дослідження роботи, є розробка архітектури обчислювальної системи на основі нейронної мережі радіально-базисних функцій, яку буде можливо упровадити як і в існуючі нейроемулятори, так і проектуємі, а також спроектувати нейрокомп’ютер, у якому буде реалізовано дану архітектуру.

Об’єкт дослідження – процеси передавання інформації, параметризації, паралельної обробки даних, оптимізації структури та показників у складі нейромереж.

Предмет дослідження – типи та архітектури нейронних мереж.

Методи досліджень. Для дослідження особливостей побудування нейронних мереж було розглянуто основні принципи роботи НМ, конструктивні особливості та специфіку роботи структури.

Наукова новизна одержаних результатів. Результати проведенного дослідження дозволяє глибше зрозуміти потенційні можливості та окреслити найбільш сприятливі сфери для використання нейронних нереж. Запропановані методи навчання штучних нейроструктур дозволяють провести оптимізацію процесу їх параметризації та забезпечити гарантоване навчання у визначений час та необхідною точністю.

Практичне значення отриманих результатів. Розглянуті дослідження роблять можливим використання Нейронних мереж у ще більших сферах, а більш детальний розгляд надає змогу виконувати операції продуктивніше у порівнянні з аналоговими комп’ютерами 5-го покоління.

Апробація роботи. Апробація досліджень відбувалась на конференції КІСМ 2022 від Криворізького національного університету.

Провідним лідером у розробці інтелектуального програмного забезпечення, що ґрунтується на засадах штучного інтелекту, є компанія Numenta, серед останніх розробок якої є програмне забезпечення, що здійснює моделювання суджень і працює за принципами людського мозку.

Програмне забезпечення Numenta працює за принципами самонавчальної штучної нейронної мережі. Топологія мережі відображує ієрархічну природу

існуючої реальності подібно до того, як це здійснює наша свідомість, постійно деталізуючи оточуючу дійсність на сукупність складових.

У теоретичних напрямах розвитку систем штучного інтелекту розрізняють дві провідні гілки, які відповідають висхідним та низхідним методам моделювання.

Згідно висхідного методу моделювання теоретичні положення ґрунтуються на основі дослідних даних нейрофізіології.

У відповідності з низхідним методом моделювання теоретичні положення ґрунтуються на відтворенні зовнішніх проявів інтелектуальної поведінки індивідуума. Цей метод моделювання, що також називається функціоналістським, є орієнтованим на широке практичне застосування, тому отримав значну фінансову і академічну підтримку та здобув значне поширення.

На сьогодні штучні нейронні мережі слугують вдалим базисом для представлення інформаційних моделей складних інженерних систем, де паралельність обчислень є одним із принципів побудови нейромережних структур.

Проектування нейрокомп’ютерів та розробка паралельних нейроалгоритмів навчання нейромереж є перспективним напрямком у ІТ технологіях. Адже час коли всі розробники процесорів гналися за збільшенням тактової частоти процесорів закінчився. Новим етапом у розвитку мікропроцесорної техніки є паралелізація обчислень. І тому почали з’являтися багатоядерні процесори різних фірм.

Нейрокомп’ютери – це обчислювальні системи шостого покоління, які в ходять до складу наукового напряму – нейрокомп’ютенгу.

За допомогою даного виду комп’ютерів, стало можливим з досить високим ступенем ефективності розв’язувати цілий ряд інтелектуальних задач, а саме: розпізнавання, образів, адаптивне управління, прогнозування, діагностика та ін..

Нейроком’ютери на відміну від комп’ютерів попередніх поколінь, відрізняються не тільки великими як наявними так і перспективними можливостями, а ще й тим, що міняється спосіб використання комп’ютера. Замість виконання програми, комп’ютер навчається. В основу навчання лягає корекція вагових зв’язків, у результаті котрого кожний вхідний сигнал (ваговий вплив) приводить до відповідного вихідного сигналу.

В основі нейрокомп’ютера стоїть нейронна мережа. Яка у свою чергу забезпечує рішення складних задач за час який дорівнює часу спрацювання електронних або оптичних елементів.

Кожна нейромережа являє собою сукупність елементів (нейронів), які з’єднані між собою як послідовно так і паралельно, що дозволяє значно швидше виконувати поставленні завдання.

На сьогоднішній час, окрім ряду переваг використання нейромереж, існує один великий недолік – час необхідний для навчання мережі, який може коливатися від 1хв, до декількох годин, в залежності від поставленого завдання, особливого коли використовується звичайний нейроемулятор (наприклад, NeuroSolution, NeuroWizard).

Розділ 1

СТАН ПИТАННЯ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Основні відомості

Загальні принципи побудови нейромереж були закладені на початку другої половини 20 століття в роботах таких учених, як: Д. Хебб, М. Мінський, Ф. Розенблат [1]. Перші нейромережі складалися з одного шару штучних нейронів - персептронов. М. Мінським було доведено ряд теорем які визначали принципи функціонування нейромереж. Незважаючи на численні переваги персептронів: лінійність, простота реалізації паралельних обчислень, оригінальний алгоритм навчання й т.п., М. Мінський разом із співавторами показав, що реалізовані на його основі одношарові нейромережі не здатні вирішити велику кількість різноманітних завдань. Це спричинило певне ослаблення темпів розвитку нейромережевих технологій в 60-ті роки. Надалі, було знято багато обмежень по використанню нейромереж з, розробкою багатошарових нейромереж, визначення яким вперше дав Ф.Розенблат: "під багатошаровою нейромережею розуміється така властивість структури перетворення, що здійснюється стандартною розімкнутою нейромережею при топологічному, а не символьному описі". У подальшому теорія нейромереж знайшла розвиток в 70-80 роках у роботах Б. Уідроу, Андерсона, Т. Кохонена, С. Гроссберга й ін. [1]

Теорія нейромереж не вносить революційних нововведень в алгоритми адаптації й оптимального керування. Оскільки системи, що самонавчаються, відомі давно, теорія адаптивних регуляторів також добре розроблена, і доволі широко застосовуються в техніці. Теорія нейромереж бере за основу розроблені раніше методи й намагається їх пристосувати для створення усе більш ефективних нейросистем. Особливу важливість використання нейроструктури здобувають з погляду продуктивності ЕОМ. Відповідно до гіпотези Мінського [2-4]: реальна продуктивність типової паралельної обчислювальної системи з n процесорів зростає як log(n) (тобто продуктивність системи з 100 процесорів усього вдвічі вище, ніж продуктивність 10-типроцесорної системи - процесори довше чекають своєї черги, чим обчислюють). Однак, якщо використовувати для рішення завдання нейромережу, то паралелізм може бути використаний практично повністю - і продуктивність зросте "майже пропорційно" n.

Нейрокомп’ютер - це обчислювальна система з MSIMD архітектурою, тобто з паралельними потоками однакових команд і множинним потоком даних. Основні переваги нейрокомп’ютерів пов'язані з масовим паралелізмом обробки, що спричиняє високу швидкодію, низькі вимоги до стабільності й точності параметрів елементарних вузлів, стійкості до перешкод і руйнувань при великій просторовій розмірності системи, причому стійкі й надійні нейросистеми можуть створюватися з низьконадійних елементів, що мають великий розкид параметрів.[5]

1.2 Біологічний нейрон

Нервова система людини побудована з елементів (нейронів), які мають приголомшуючу складність. Близько 10^{11} нейронів беруть участь в приблизно 10^{15} передаючих зв'язках, що мають довжину метр і більше. Кожен нейрон володіє багатьма якостями, що є спільними з іншими елементами тіла, але його унікальною здатністю є прийом, обробка і передача електрохімічних сигналів по нервових шляхах, які утворюють комунікаційну систему мозку.

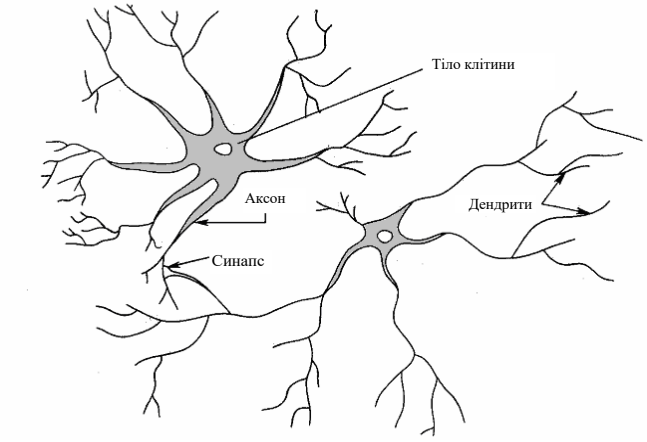


Рисунок 1.1 - Біологічний нейрон

На рис. 1.1 показана структура пари типових біологічних нейронів. Дендрити (входи нейрона) йдуть від тіла нервової клітини до інших нейронів, де вони приймають сигнали в точках з'єднання (синапсах). Прийняті синапсом вхідні сигнали підводяться до тіла нейрона. Тут вони підсумовуються, причому одні входи стимулюють активізацію нейрона, а інші – зниження його активності. Коли сумарна активність (збудження) нейрона перевищує деякий поріг, нейрон переходить в активний стан, посилаючи по аксону (виходу нейрона) сигнал іншим нейронам. У цієї основної функціональної схеми багато спрощень і виключень, проте більшість штучних нейронних мереж моделює лише ці прості властивості.

1.3 Штучний нейрон

Основними компонентами нейромережі є нейрони /neurons/ (елементи, вузли), які з'єднані зв'язками. Сигнали передаються по зваженим зв'язкам (connection), з кожним з яких пов'язаний ваговий коефіцієнт (weighting coefficient) або вага.

Серед моделей НМ – програмні і апаратні, найбільш поширені – програмні.

Цілями використання є розпізнавання образів, прогнозування, створення асоціативної пам'яті.

Штучний нейрон імітує в першому наближенні властивості біологічного нейрона. На вхід штучного нейрона поступає множина сигналів, які є виходами інших нейронів. Кожен вхід множиться на відповідну вагу, аналогічну його синаптичній силі, а всі виходи підсумовуються, таким чином визначаючи рівень активації нейрона. На рис.2 представлена модель, що реалізує цю ідею. Хоча мережеві парадигми досить різноманітні, в основі майже всіх їх лежить ця конфігурація. Можна побачити як множина вхідних сигналів, що позначені як x_1, x_2,…, x_n, поступають на штучний нейрон. Ці вхідні сигнали, котрі в сукупності позначаються як вектор X, схожі на сигнали, що приходять в синапси біологічного нейрона. Кожен сигнал множиться на відповідну вагу w_1, w_2,…, w_n, і поступає в підсумовуючий блок, що позначений як Σ. Кожна вага відповідає «силі» одного біологічного синаптичного зв'язку (множина ваг в сукупності позначається як вектор W). Підсумовуючий блок, який відповідає тілу біологічного нейрона, складає зважені входи алгебраїчно, створюючи вихід, який ми називатимемо NET. У векторних позначеннях це може бути компактно записано таким чином:

$$NET = XW \tag{1.1}$$

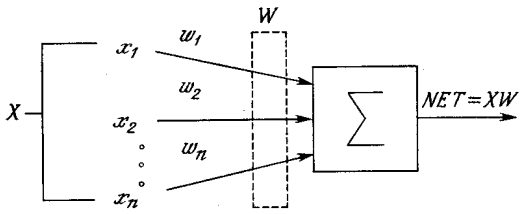


Рисунок 1.2 - Штучний нейрон

Сигнал NET далі, як правило, перетворюється активаційною функцією F і дає вихідний нейронний сигнал OUT = F(NET). Активаційна функція F(NET) може бути:

1. Пороговою бінарною функцією:

$$OUT = \begin{cases} 1, NET \geq T \\ 0, NET < T \end{cases}, \tag{1.2}$$

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

2. Лінійною обмеженою функцією:

$$OUT = \begin{cases} 1, NET \geq T \\ NET, 0 \leq NET < T \\ 0, NET < 0 \end{cases}, \quad (1.3)$$

3.Функцією гіперболічного тангенса:

$$\text{OUT} = th(C \cdot NET) = \frac{\exp(C \cdot NET) - \exp(-C \cdot NET)}{\exp(C \cdot NET) + \exp(-C \cdot NET)}, \quad (1.4).$$

де C > 0 – коефіцієнт ширини сигмоїди по осі абсцис (звичайно C=1).

4. Сигмоїдною (S-подібною) або логістичною функцією:

$$\text{OUT} = \frac{1}{1 + e^{-C \cdot \text{NET}}}, \quad (1.5)$$

З виразу для сигмоїда очевидно, що вихідне значення нейрона лежить в діапазоні [0,1] (рис.4). Популярність сигмоїдної функції зумовлюють наступні її властивості:

- здатність підсилювати слабкі сигнали сильніше, ніж великі, і опиратися „насиченню” від потужних сигналів;
- монотонність і диференційованість на всій осі абсцис;
- простий вираз для похідної.

1.4 Одношарові нейронні мережі

Хоча один нейрон здатний виконувати прості процедури розпізнавання, сила нейронних обчислень виникає від з'єднань нейронів в мережах. Проста мережа складається з групи нейронів, що створюють шар, як показано в правій частині рис.6. Відзначимо, що вершини-круги зліва служать лише для розподілу вхідних сигналів. Вони не виконують будь-яких обчислень, і тому не вважатимуться шаром. З цієї причини вони позначені кругами, щоб відрізняти їх від обчислюючих нейронів, які позначені квадратами. Кожен елемент з множини входів X сполучений окремою вагою з кожним штучним нейроном. Кожен нейрон видає зважену суму входів в мережу. У штучних і біологічних мережах багато з'єднань можуть бути відсутніми, можуть мати місце також з'єднання між виходами і входами елементів в шарі.

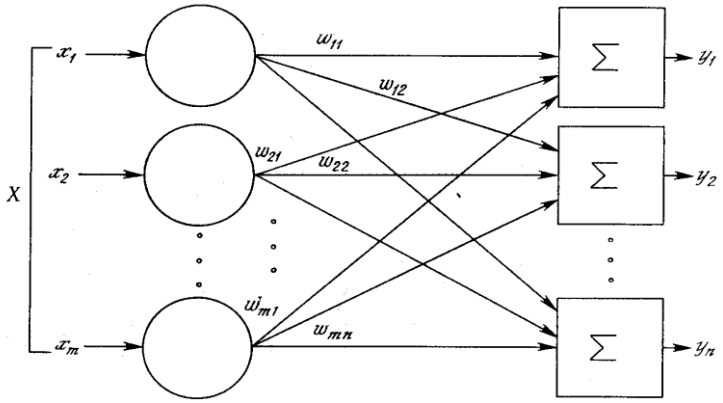


Рисунок 1.3 - Одношарова НМ

Зручно вважати вагу елементами матриці W. Матриця має m рядків і n стовпців, де m - число входів, а n - число нейронів. Наприклад, $w_{3,2}$ - це вага, що пов'язує третій вхід з другим нейроном. Таким чином, обчислення вихідного вектора Y, компонентами якого є виходи OUT нейронів, зводиться до матричного множення Y = XW, де Y і X - вектори-рядки.

Як науковий предмет штучні нейронні мережі вперше заявили про себе в 40-ві роки. Прагнучи відтворити функції людського мозку, дослідники створили прості апаратні (а пізніше програмні) моделі біологічного нейрона та системи його з'єднань. Коли нейрофізіологи досягли глибшого розуміння нервової системи людини, ці ранні спроби стали сприйматися як досить грубі апроксимації. Проте на цьому шляху були досягнуті вражаючі результати, що стимулювали подальші дослідження, що привели до створення досконаліших мереж.

Перше систематичне вивчення штучних нейронних мереж було зроблене Мак-Каллоком і Піттсом в 1943 р.. В роботі вони досліджували мережеві парадигми для розпізнавання зображень, що піддаються зсувам і поворотам. Проста нейронна модель, показана на рис.1.4, використовувалася в більшій частині їх роботи. Елемент Σ множить кожен вхід x на вагу w і підсумовує зважені входи. Якщо ця сума більше заданого порогового значення, вихід рівний одиниці, інакше - нулю. Ці системи (і подібних ним множинам) одержали назву персептронів. Вони складаються з одного шару штучних нейронів, сполучених між собою за допомогою вагових коефіцієнтів з множиною входів (рис.1.5), хоча описуються і більш складніші системи.

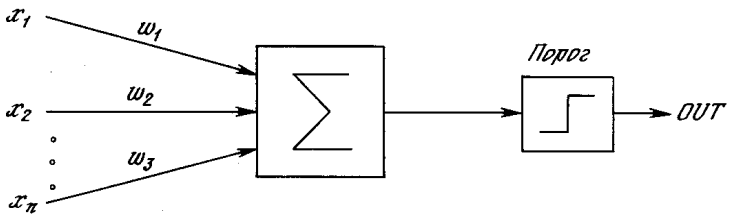


Рисунок 1.4 - Нейрон персептрона

Перші персептрони були створені Ф.Розенблатом у 60-х роках і викликали великий інтерес. Первинна ейфорія змінилася розчаруванням, коли виявилося, що персептрони не здатні навчитися вирішенню ряду простих задач. М.Мінський строго проаналізував цю проблему і показав, що існують жорсткі обмеження на те, що можуть виконувати одношарові персептрони, а отже і на те, чому вони можуть навчатися. Оскільки у той час методи навчання багатошарових мереж ще не були відомі, дослідники перейшли в більш багатообіцяючі області, що в свою чергу призвело дослідження у області нейронних мереж до занепад. Відкриття методів навчання багатошарових мереж, більш ніж який-небудь інший чинник, вплинуло на відродження інтересу і дослідницьких зусиль.

Навчання персептрона виглядає так:

1. Ініціалізація вагових матриць W (випадкові значення)
2. Подати вхід X і обчислити вихід Y для цільового вектора Y^T
3. Якщо вихід правильний – перейти на крок 4; інакше обчислити різницю $D = Y^T – Y$; модифікувати ваги за формулою:

$$w_{ij}(e+1) = w_{ij}(e) + \alpha D X_i, \qquad (1.6)$$

де $w_{ij}(e)$ - значення ваги від нейрона і до нейрона j до налагодження, $w_{ij}(e+1)$ - значення ваги після налагодження, α - коефіцієнт швидкості навчання, X_i - вхід нейрона і, e – номер епохи (ітерації під час навчання).

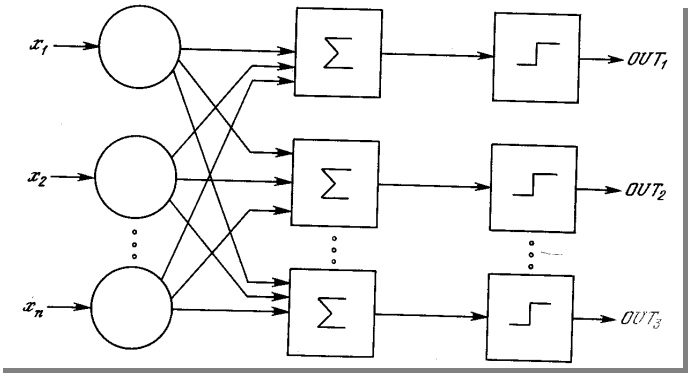


Рисунок 1.5 – Персептрон

4. Виконувати цикл з кроку 2, поки мережа не перестане помилятися. На другому кроці у випадковому порядку пред’являються всі вхідні вектори.

Один з найпесимістичніших результатів Мінського показує, що одношаровий персептрон не може відтворити таку просту функцію, як ВИКЛЮЧАЄ АБО. Це функція від двох аргументів, кожний з яких може бути нулем або одиницею. Вона приймає значення одиниці, коли один з аргументів рівний одиниці (але не обидва).

1.5 Багатошарові нейронні мережі

Багатошарові мережі (1.6) мають значно більше можливостей, ніж одношарові. Проте багатошарові мережі в свою чергу можуть привести до збільшення обчислювальної потужності, в порівнянні з одношаровими лише в тому випадку, якщо активаційна функція між шарами буде нелінійною.

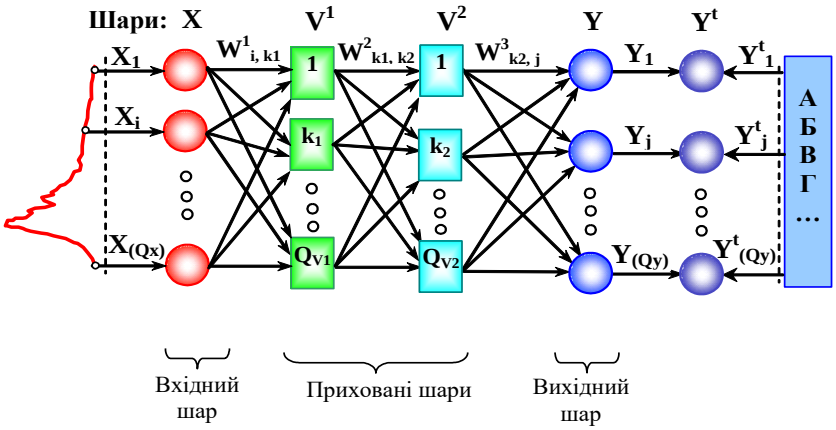


Рисунок 1.6 - Багатошарова НМ

Обчислення виходу шару полягає в множенні вхідного вектора на першу вагову матрицю з подальшим множенням (якщо відсутня нелінійна активаційна функція) результуючого вектора на другу вагову матрицю $(XW_1)W_2$. Оскільки множення матриць асоціативне, то $X(W_1W_2)$. Це показує нам, що двошарова лінійна мережа еквівалентна одному шару з ваговою матрицею, що рівна добутку двох вагових матриць. Отже, для лінійної активаційної функції, будь-яка багатошарова лінійна мережа може бути замінена еквівалентною одношаровою мережею.

1.6 Архітектура нейрокомп’ютерів

1.6.1 Апаратна реалізація нейрообчислювачів

На сьогодні можна виділити три основні напрямки розвитку обчислювальних систем з масовим паралелізмом (ОСМП):

- ОСМП на базі каскадного з'єднання універсальних SISD, SIMD, MISD мікропроцесорів : елементна база - універсальні RISC або CISC процесори: Intel, AMD, Sparc, Alpha, Power PC, MIPS та ін.;
- ОСМП на базі процесорів з розпаралелюванням на апаратному рівні: елементна база - DSP процесори: TMS, ADSP, Motorola, ПЛІС;
- ОСМП на спеціалізованій елементній базі - елементна база від спеціалізованих однобітних процесорів до нейрочипів.

Нейромережеві системи, які реалізовані на апаратних платформах першого напрямку можна відносити до нейроемуляторів - тобто систем які реалізовують типові нейрооперації (зважене підсумовування й нелінійне перетворення) на програмному рівні[3]. Нейромережеві системи, другого й третього напрямку реалізовуються у вигляді плат розширення стандартних обчислювальних систем (1-го напрямку) - називаються нейроприскорювачами, а системи, реалізовані на апаратній платформі третього напрямку у вигляді функціонально закінчених обчислювальних пристроїв, варто відносити до нейрокомп'ютерів (всі операції виконуються в нейромережевому логічному базисі).

Нейроприскорювачі можна розділити на два класи: "віртуальні", які вставляються до слоту розширення стандартного PC, та "зовнішні", що з'єднуються з керуючою ЕОМ по конкретному інтерфейсу або шині [6].

Побудова нейроприскорювачів на базі ПЛІС, з одного боку, дозволяє гнучко реалізувати різні нейромережеві парадигми, а з іншої сполучено з великою проблемою розведення всіх необхідних міжз'єднань (рис.1.1). Сьогодні, ПЛІС мають різні функціональні можливості (число вентилів від 5 до 100 тисяч). Нейрообчислювачі на базі ПЛІС, як правило, позиціонуються як гнучкі нейрообчислювальні системи для науково-дослідних цілей і дрібносерійного виробництва. Для побудови більш продуктивних і ефективних нейрообчислювачів, як правило, потрібне застосування сигнальних процесорів.

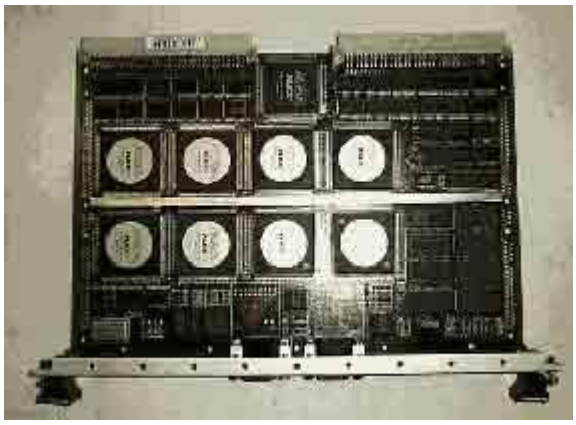


Рисунок 1.7 - Зовнішній вигляд нейроприскорювача ППО

Загальна структурна схема нейроприскорювача представлена на рис.1.8. До складу даного НП входять:

- схема управління (Сх Упр);
- базові обчислювальні елементи (БВЭ1-БВЭ6);
- контролер зовнішньої шини (Контролер E-bus);
- контролер системної шини (Контролер VME);
- два масиви статичної пам'яті (ОЗУ0, ОЗУ1);
- блок високошвидкісних приймачів/передатчиків;

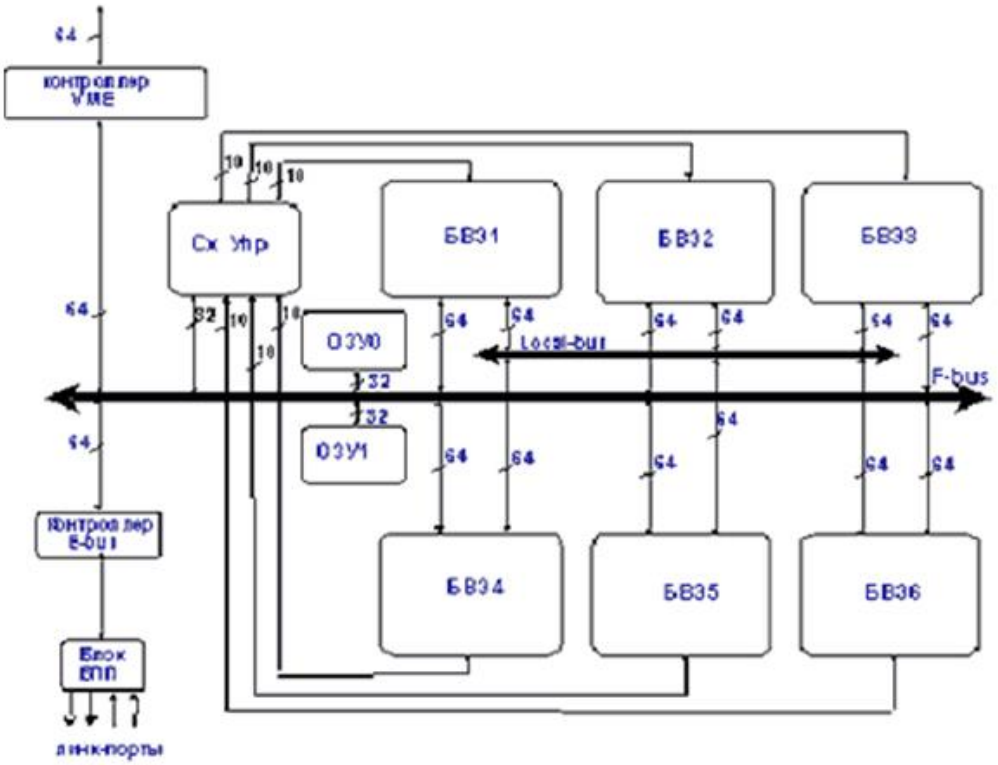


Рисунок 1.8. - Загальна структура нейроприскорювача

Нейроприскорювачі на базі каскадного з'єднання сигнальних процесорів. Такі нейрообчислювачі представляють собою системи з можливістю паралельної обробки даних, що в свою чергу дозволяє реалізовувати на їх основі нейрообчислювальні системи, у структурі яких можна виділити дві основні частини :

– керуючу Host-ЕОМ, реалізовану на основі звичайної обчислювальної системи з CISC- або RiSC- мікропроцесорами;

– віртуальний (або зовнішній) апаратний засіб, що підключається до Host-ЕОМ за допомогою внутрішніх (зовнішніх) системних інтерфейсів, що виконують основні обчислювальні операції.

Загальну функціональну схему НО представлено на рис. 1.9. В основі побудови нейрообчислювачі даного типу лежить використання сигнальних процесорів, об'єднаних між собою відповідно до певної архітектури, що в свою чергу забезпечує паралельність виконання обчислювальних операцій. Як правило, такі НО будуються на основі гнучкої модульної архітектури, що забезпечує простоту конфігурації системи й збільшення обчислювальної потужності, шляхом збільшення числа процесорних модулів або застосування більше продуктивних сигнальних процесорів. НО даного типу реалізуються в основному на базі несучих модулів стандартів ISA, PCI, VME. Основними їх функціональними елементами є модуль матричних сигнальних процесорів (МСП), робоча пам'ять, пам'ять програм, модуль забезпечення уведення

висновку сигналів (включаючий в себе АЦП, ЦАП і TTL лінії), а також модуль керування, що може бути реалізований на основі спеціалізованого керуючого сигнального процесора (КП), на основі ПЛІС або мати розподілену структуру, при якій функції загального керування розподілені між МСП [5-6].

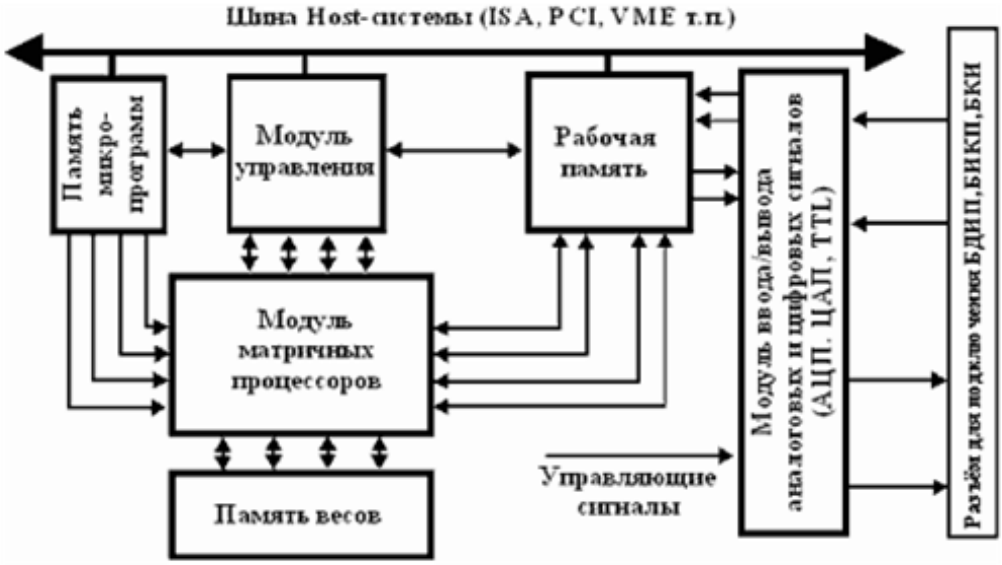


Рисунок 1.9 - Загальна схема нейрообчислювача

1.6.2 Елементна база нейрообчислювачів

Елементною базою НО систем другого й третього є відповідно трансп'ютери, цифрові сигнальні процесори (DSP), ПЛІС та нейрочіпи. Причому використання, як тих, так і інших, дозволяє сьогодні реалізовувати НО, котрі функціонують у реальному масштабі часу.

Елементною базою перспективних НО є нейрочіпи. Їх виробництво ведеться в багатьох країнах світу, причому більшість із них на сьогодні орієнтована на закрите використання (тобто створювалися для конкретних спеціалізованих керуючих систем). [7]

Всі нейрочіпи класифікуються наступним чином:

- по типу логіки поділяються на: цифрові, аналогові й гібридні;
- по типу реалізації нейроалгоритмів: з повністю апаратною реалізацією та з програмно-апаратною реалізацією (коли нейроалгоритми зберігаються в ПЗУ);
- по характеру реалізації нелінійних перетворень: на нейрочіпи із твердою структурою нейронів (апаратно реалізованих);

– по можливостям побудови нейромереж: нейрочіпи із твердою та змінною нейромережевою структурою (тобто нейрочіпи у яких топологія нейромереж реалізована жорстко або гнучко).

Процесорні матриці - це чіпи, які близькі до звичайних RISC процесорів і котрі об'єднують у своєму складі деяке число процесорних елементів, вся ж інша логіка, як правило, повинна бути реалізована на базі периферійних схем.

В окремий клас варто виділити так звані нейросигнальні процесори, ядро яких являє собою типовий сигнальний процесор, а реалізована на кристалі додаткова логіка забезпечує виконання нейромережевих операцій (наприклад, додатковий векторний процесор і т.п.).

1.6.3 Нейрокомп’ютери

Розглядаючи підходи до апаратної реалізації нейрообчислювачів необхідно відмітити те, що не дивлячись на широке поширення різних високопараллельних прискорювачів для різних завдань, число моделей повнофункціональних нейрокомп’ютерів невелике, а комерційні доступні з них одиниці[2]. Це викликано тим що більшість із них реалізовані для спецзавдань. Найбільш яскравим прикладом нейрокомп’ютерів є: Нейрокомп’ютер Synaps 1 (Siemens, Германія), нейрокомп’ютер "Силіконовий мозок" (створений у США по програмі "Електронний мозок", призначений для обробки аерокосмічних зображень, продуктивність 80 петафлоп (80х1015 операцій у секунду, обсяг дорівнює обсягу мозку людини, споживаюча потужність 20 Вт.).

Нейрокомп’ютер Synaps 1. Базовий комплект SYNAPSE1-N110 припускає наявність головної ЕОМ - робочої станції SUN SPARCSTATION 5 моделі TX1 у якості допоміжного консолідуючого пристрою, що полегшує процеси програмування, проектування нейромереж, тестування, керування зовнішніми пристроями, висновку результатів і т.п. (Рис.1) Головна ЕОМ сполучається з апаратурою SYNAPSE1 через шину VME. В архітектурі SYNAPS1 можна виділити чотири основні компоненти: матричний процесор, пам'ять ваг, пристрій керування та пристрій даних з наступними характеристиками:

– процесорна плата з матрицею з 8-ми сигнальних процесорів MA 16 із продуктивністю 3,2 мільярди операцій множення (16х16 біт) і додавання (48 біт) у секунду
– пам'ять ваг 128Мб
– пристрій керування на базі Моторолла MC68040
– пристрій даних на базі Моторолла MC68040
– псі апаратні засоби розміщаються в невеликому корпусі 667х398х680 мм.

Дослідження показали, що по продуктивності виконання нейромережевих операцій, на нейрокомп’ютері Synaps 1, принаймні на три порядки перевищують продуктивність традиційних обчислювальних систем і дозволяють моделювати нейромережі з кількістю синапсів рівним 64 млн, а гнучкість архітектури практично не обмежує різноманітність реалізованих нейромережевих парадигм[8].

Рисунок 1.10- Зовнішній вигляд нейрокомп’ютера Synapse1

Нейрокомп’ютер «Ембріон». Даний комп’ютер є моделлю мозку людини, збудження квазінейронів у ньому створює віртуальне квантове, когерентне, хвильове поле, яке має відношення до проблем створення квантового нейронного комп’ютера (рис. 1.11)

Рисунок 1.11 - Зовнішній вигляд нейрокомп’ютера «Ембріон»

Даний НК представляє собою деякий різновид квантової Макросистеми або квантово-механічної системи і потребує для свого створення зовсім іншої функціональної елементної бази[8].

1.7 Огляд найпоширеніших нейроалгоритмів навчання

1.7.1 Основні відомості

Штучні нейронні мережі навчаються найрізноманітнішими методами. Основною ознакою більшості методів навчання є те, що вони виходять із загальних передумов і мають багато ідентичних характеристик.

Навчальні алгоритми можуть бути класифіковані як алгоритми навчання із учителем і навчання без учителя. У першому випадку існує вчитель, що пред'являє вхідні образи мережі, порівнює результуючі виходи з необхідними, а потім будує ваги мережі таким чином, щоб зменшити розходження. Важко представити такий навчальний механізм у біологічних система. Отже, хоча даний підхід привів до більших успіхів при рішенні прикладних завдань, він відкидається тими дослідниками, котрі думають, що штучні нейронні мережі обов'язково повинні використовувати ті ж механізми, що й людський мозок

У другому випадку навчання без учителя: при пред'явленні вхідних образів мережа самоорганізується, набудовуючи свої ваги відповідно до певного алгоритму. Необхідний вихід у процесі навчання не зазначений і саме тому результати визначення збудливих образів для конкретних нейронів непередбачені. При цьому, однак, мережа організується у формі, що відбиває істотні характеристики навчального набору. Наприклад, вхідні образи можуть бути класифіковані відповідно до ступеня їхньої подібності так, що образи одного класу активізують той самий вихідний нейрон.

1.7.2 Алгоритми навчання з учителем

Алгоритм навчання персептрону. В 1957 р. Р.Розенблатт розробив модель, що викликала великий інтерес у дослідників [5] . Незважаючи на деякі обмеження її вихідної форми, вона стала основою для багатьох сучасних, найбільш складних алгоритмів навчання із учителем. Даний алгоритм є алгоритмом прямого розповсюдження.

Персептрон є дворівневої нерекурентною мережею. Вона використовує алгоритм навчання із учителем, інакше кажучи, навчальна вибірка складається з безлічі вхідних векторів, для кожного з яких зазначений свій необхідний вектор мети. Компоненти вхідного вектора представлені безперервним діапазоном значень; компоненти вектору результату є бінарними величинами (0 або 1). Після навчання мережа одержує на вході набір безперервних входів і виробляє необхідний вихід у вигляді вектора з бінарними компонентами.

Алгоритм навчання має наступні кроки:

a. всі ваги мережі перетворюються у малі величини випадковим чином;
b. на вхід мережі подається вхідний навчальний вектор X і обчислюється сигнал N від кожного нейрона, використовуючи наступну формулу:

$$N_j = \sum_i W_{ij} * x_i \quad (1.6)$$

c. обчислюється значення граничної функції активації для сигналу N від кожного нейрона в такий спосіб:

$$Y_j = \begin{cases} 1, якщо N_j > \theta_j \\ 0, якщо N_j < \theta_j \end{cases} \quad (1.7)$$

d. обчислюється помилка для кожного нейрона за допомогою вирахування отриманого виходу з необхідного виходу:

$$ERROR_j = TAR_j - Y_j \quad (1.8)$$

e. кожна вага модифікується наступним чином:

$$W_{ij}(t+1) = W_{ij}(t) + \alpha * x_i * ERROR_j \quad (1.9)$$

f. кроки 2-5 повторюються до тих пір поки, значення помилки ERROR не стане таким, як його задали на початку.

Область використання алгоритму: розпізнавання образів та класифікація.

Недоліки: примітивні відокремлюючи поверхні (гіперповехні) в свою чергу дають можливість розв’язувати лише найпростіші задачі розпізнавання.

Переваги: програмні та апаратні реалізації моделі досить прості, а отже можна сказати що даний алгоритм є простий та швидкий.

Алгоритм навчання Уідроу-Хоффа. Недоліком алгоритму навчання персептрону є те, що він обмежується бінарними виходами[5]. Алгоритм навчання Уідроу-Хоффа є розширеним у випадку безперервних виходів, використовуючи сигмоїдальну функцію. Модифікований крок 4 у цьому випадку реалізується в такий спосіб:

d. обчислення помилки для кожного нейрона за допомогою вирахування отриманого виходу з необхідного виходу $ERROR_j = TAR_j - N_j$

Алгоритм статичного навчання. Алгоритм статичного навчання являє собою таким алгоритмом, який забезпечує навчання багатошарових нейромереж.

Існує багато варіацій на тему статистичного навчання. Наприклад, глобальна енергія може бути визначена як середня квадратична помилка між отриманим і бажаним вихідним вектором з того, якого навчають, безлічі, а змінними можуть бути ваги мережі. У цьому випадку мережа може бути навчена, починаючи з високої штучної температури, шляхом виконання наступних кроків:

- подати навчальний вектор на вхід мережі й відповідно до відповідних мережних правил обчислити вихід;
- обчислити значення середньої квадратичної помилки між бажаними й отриманим вихідними векторами;
- змінити мережні ваги випадковим образом, потім обчислити новий вихід і результуючу помилку. Якщо помилка зменшилася, залишити змінену вагу; якщо помилка збільшилася, залишити змінену вагу з імовірністю, обумовленої розподілом Больцмана. Якщо зміни ваг не виробляється, то повернути вагу до його попереднього значення;
- повторити кроки з 1 по 3, поступово зменшуючи вхідні значення.

Алгоритм зворотного розповсюдження помилки (Back Propagation). Зворотне розповсюдження це систематичний метод для навчання багатошарових штучних нейронних мереж[6]. Не дивлячись на деякі ліміти, дана процедура досить сильно розширила область проблем, в яких можуть бути використанні нейронні мережі.

Навчання мережі відбувається після виконання наступних етапів:

- вибір чергової навчаючої пари та подання її на вхід мережі;
- обчислення виходу мережі;
- обчислення різниці між виходом мережі та цільовим виходом;
- корекція помилки;
- повторення попередніх кроків, доти доки помилка не стане заданою.

Даний алгоритм був широко використаний у сфері прикладних дослідів. Фірма NEC у Японії заявила, що зворотне розповсюдження було нею використано для візуального розпізнавання букв, причому точність перевищила 99%. Даний результат був досягнутий в результаті комбінації звичайних алгоритмів з мережею зворотного розповсюдження, що забезпечують додаткову перевірку. Зворотне розповсюдження також використовувалося у машинному розпізнаванні англійських слів. Літери, нормалізовані по розміру, наносились на сітку. Дані проекції слугували входами для мереж ВР. Точність перевищувала 97%.

Область використання: розпізнавання образів, класифікація та прогнозування.

Недоліки: багатокритеріальна задача оптимізації в даному алгоритмі розглядається як набір однокритеріальних задач, тобто на кожній ітерації відбуваються зміни значень параметрів мережі, котрі покращують роботу мережі лише з одним прикладом навчаючої вибірки, а отже такий підхід зменшує швидкість навчання.

Переваги: зворотне розповсюдження є ефективним та популярним алгоритмом навчання багатошарових нейронних мереж, за допомогою якого на даний момент, розв’язується велика кількість задач.

1.7.3 Алгоритми навчання без учителя

Алгоритм навчання Хебба. Суть даного алгоритму полягає у тому, що синоптичні з'єднання двох нейронів підсилюється, якщо обидва нейрона збуджені[6]. Це можна представити як посилення синапсу відповідно до кореляції рівнів збуджених нейронів, що з'єднуються даним синапсом. Тому алгоритм навчання Хэбба іноді називається кореляційним алгоритмом.

Ідея алгоритму полягає у наступному:

$W_{ij}(t+1) = W_{ij}(t) + Ni * Nj$, де $W_{ij}(t)$ сила синапсу від нейрона і до нейрона j в момент часу t; N_i - рівень порушення передсинаптичного нейрона; N_j - рівень порушення постсинаптического нейрона.

У методі Хебба навчання є винятково локальним явищем, що охоплює тільки два нейрони та з'єднуючий їх синапс; не потрібно використання глобальної системи зворотного зв'язку для розвитку нейронних утворень.

Наступне використання методу Хебба для навчання нейронних мереж привело до більших успіхів, але поряд із цим показало обмеженість методу; деякі образи просто не можуть використовуватися для навчання за цим методом. У результаті з'явилася велика кількість розширень і нововведень, більшість із яких у значній мірі засновано на роботі Хебба.

Алгоритм навчання Кахонена. У основі даного алгоритмі полягає підлаштування синапсів на основі їх значень від попередніх ітерацій[5]:

$$W_{ij}(t) = W_{ij}(t-1) + \alpha[y_i^{(n-1)} - w_{ij}(t-1)] \qquad (1.10)$$

З даної формули видно, що навчання зводиться до мінімізації різниці між вхідними сигналами нейрону, котрі поступають з виходів нейронів попереднього слою ($y_i^{(n-1)}$), та ваговими коефіцієнтами його синапсів. Даний алгоритм навчання має приблизно таку ж структуру, як і алгоритмі Хебба, але на кроці обчислення вагових коефіцієнтів обирається нейрон, значення синапсів якого максимально походять на вхідний образ, і підлаштування вагових коефіцієнтів здійснюється лише по вище зазначеній формулі. Дані дії можуть супроводжуватися гальмуванням всіх інших нейронів даного слою та введенням даного нейрону у стан насищення. Вибір такого нейрону може здійснюватися, наприклад, розрахунком скалярного добутку вагових коефіцієнтів з вектором вхідних значень.

Область використання: кластерний аналіз, розпізнавання образів та класифікація.

Недоліки: мережа може використовуватися для кластерного аналізу лише у тому випадку, якщо попередньо відома кількість кластерів.

Переваги: дана мережа здатна функціонувати в умовах перешкод, так як кількість кластерів фіксована, ваги модифікуються повільно, налаштування ваг відбувається після навчання.

1.7.4 Обґрунтування обраного напряму проектування та постановка завдання

На сьогодні, штучні нейронні мережі слугують вдалим базисом для представлення інформаційних моделей складних інженерних систем, де паралельність обчислень є одним із принципів побудови нейромережевих структур.

Проектування архітектури обчислювальної системи на основі використання нейромережевих систем та використання сучасних нейроалгоритмів навчання нейромережі є перспективним напрямком у IT технологіях. Адже закінчився час коли всі розробники процесорів гналися за збільшенням тактової частоти процесорів. Новим етапом у розвитку мікропроцесорної техніки є паралелізація обчислень. І тому почали з’являтися багатоядерні процесори різних фірм. Але при всій своїй продуктивності багатоядерні комп’ютери уступають нейрокомп’ютерам (комп’ютери шостого покоління) у наступному:

- паралельна робота досить великої кількості простих обчислювальних елементів забезпечує велику продуктивність;
- нейронна мережа здатна до навчання, яке відбувається за допомогою налаштування параметрів мережі;
- проста будова окремих нейронів дозволяє використовувати нові фізичні принципи обробки інформації для апаратних реалізації нейронних мереж;
- велика перешкодо- та відмово- стійкість нейромереж;

У цілому, нейромережі дають можливість краще зрозуміти організацію нервової системи людини та тварин на середніх рівнях: пам’ять, обробка сенсорної інформації, моторика. Нейромережі використовуються для розв’язання наступних класів задач:

- нелінійна апроксимація багатомірних функцій;
- задачі прогнозування у часі для процесів, які залежать від багатьох змінних;

- задачі класифікації по багатьом ознакам, які дозволяють розбивати вхідний простір на області;
- розпізнавання образів;
- пошук по асоціаціям;
- модель для пошуку закономірностей у масивах даних;

Нейронні системи здобули високу популярність не завдяки класу розв'язуваних задач, а завдяки наступним властивостям:

- Здатність до навчання. Обираючи певну модель НМ та нейроалгоритму, ми можемо навчити мережу розв'язувати певну задачу;
- Здатність до узагальнення. Після навчання мережа стає непохитною відносно незначних змін вхідних сигналів, а отже дає правильний результат на виході;
- Здатність до абстрагування. Якщо мережі дати декілька видозмінених варіантів вхідного образу, то система сама створить на виході ідеальний образ, з котрим вона ніколи не зустрічалась.

Але з дуже великою кількістю переваг, всі нейронні мережі мають один недолік, а саме час навчання , котрий, від складності задачі та кількості даних, може істотно збільшуватись від декількох хвилин до декількох годин.

Отже, до головних завдань дипломної роботи можна віднести наступні:

- аналіз нейронної мережі радіально-базисних функцій;
- проектування обчислювальної системи для можливості функціонування нейронної мережі типу радіально-базисних функцій;
- аналіз отриманих розробок.

Висновки до розділу:

Обчислювальна система на основі використання принципу роботи «нейромережа-нейрокомп'ютер» - може бути представлений не тільки у вигляді окремого «системного блоку», як звичайний ПК, а й у вигляді плати розширення – нейрообчислювачі та нейроприскорювачі, які під'єднуються до плати розширення звичайного ПК, наприклад до шин PCI, USB, ISA, LPT та ін.. Апаратна реалізація нейрокомп'ютера має свої недоліки, а саме дороговизна розробки, та неможливість реалізації окремих її елементів. Найбільш відомим нейрокомп'ютером є Synaps1.

Серед найбільш поширених нейроалгоритмів виділяють алгоритми навчання персептрону, статичного навчання, Уідроу-Хоффа, ВР. В основі кожного з цих алгоритмів полягає сума добутків, а це означає що будь-який з вище зазначених алгоритмів можна, наприклад розпаралелити.

Розділ 2

ПРОЕКТНА ЧАСТИНА

					КНУ.РБ.123.16.35.02.ПЧ			
Змн.	Арк.	№ документа	Підпис	Дата				
Розробив		Лебідь			ПРОЕКТНА ЧАСТИНА	Літера	Аркуш	Аркушів
Перевірив		Купін						
						КІ-21М		
Н.контроль		Кузнєцов						
Затвердив		Купін						

2.2 Вибір елементної бази для обчислювальної системи

2.2.1 Аналіз існуючих рішень

Елементною базою сучасних нейрокомп’ютерів є трансп’ютери, цифрові сигнальні процесори (DSP), ПЛІС та нейрочіпи. Причому використання, як тих, так і інших, дозволяє сьогодні реалізовувати нейрообчислювачі, що функціонують у реальному масштабі часу. Найперспективнішою та найпопулярнішою на сьогоднішній день елементною базою для нейрокомп’ютерів є нейрочіпи.

Для оцінки продуктивності нейрокомп’ютерів та нейрообчислювачів використовуються наступні показники:

- CUPS (connections update per second) - число змінених значень ваг у секунду (оцінює швидкість навчання);
- CPS (connections per second) - число з'єднань (множень із нагромадженням) у секунду (оцінює продуктивність);
- CPSPW = CPS/Nw, де Nw - число синапсів у нейроні;
- CPPS - число з'єднань примітивів у секунду, CPPS=CPS*Bw*Bs, де Bw, Bs - розрядність ваг і синапсів;
- MMAC - мільйонів множень із нагромадженням у секунду.

Розробка нейрочипів ведеться в багатьох країнах світу. На сьогодні виділяють дві базові лінії розвитку обчислювальних систем з масовим паралелізмом: ВСМП із модифікованими послідовними алгоритмами, характерними для однопроцесорних фоннеймановських алгоритмів і ВСМП на основі принципово нових супер паралельних нейромережевих алгоритмів рішення різних завдань [2].

Характеристики найбільш поширених нейрочипів приведені у табл. 2.1. , де максимальна кількість синапсів означає максимальне число, що визначає розмір внутрішньо кристальної пам'яті ваг, максимальна кількість [11].

Таблиця 2.1 -Характеристики найбільш поширених нейрочипів

Наименование	Фирма изготовитель	Разряд-ность, бит	Максимальное количество синапсов*	Максимальное число слоев**	Примечание
MA16	Siemens	48 (умножители и сумматоры)	-	-	400 MMAC.
NNP (Neural Networks Processor)	Accurate Automation	Nx16	-	-	MIMD, N - число процессоров.
CNAPS-1064	Adaptive Solutions	16	128 Кбайт	64	
100 NAP Chip	HNC	32	512 Кбайт	4	Плав. Арифм. 4 процессорных элемента
Neuro Matrix NM6403, Такт. частота 50 МГц.	Модуль, Россия	64 (вект. процессор), 32 RISC ядро	4096 шт.	24	Совместим с портами TMS320C4x
Neuro Matrix NM6404, Такт. частота 133 МГц.	Модуль, Россия	64 (вект. процессор), 32 RISC ядро	4096 шт.	~48	Совместим с портами TMS320C4x
CLNN 32 CLNN 64	Bellcore	32 64	496 1024	32 нейрона	10^8 перекл./с 2 x 10^8 перекл./с
NC 3001	NeuriGam	16	4096 шт.	32	
ZISC 036 (Zero Instruction Set Computer)	IBM	64 разр. входного вектора	-	36 нейронов	Частота 20МГц, Векторно-прототипный нейрочип
ETANN 80170NW	Intel	64 входа	Два банка весов 64x80	64 нейрона в слое, 3 слоя.	Аналоговая
MD-1220	Micro Devices	16	64 шт.	8	8 нейронов
MT 19003 - Neural Instruction Set Processor	Micro Circuit Engineering (MCE)	16 разр. Умножитель 35 разр. сумматор	-	1	RISC МП с 7 специальными командами
Neuro Fuzzu	National Semiconductor	-	-	-	
NI 1000	Nestor	5-16 (одного нейрона)	-	1024 прототипных 256 мерных векторов	Векторно-прототипный нейрочип
NLX420 (NLX 110, 230)	Adaptive Logic	16	1 Мбайт	16	16 процессорных элементов
OBL Chip	Oxford Computer	16	16 Мбайт	-	
L-Neuro 1.0 L-Neuro 2.3	Philips	16 16	1536	16 нейронов 192 (12x16)	26 МГц 60 МГц
RSC (Speech Recognition Chip) - 164	Sensory Circuits	-	-	-	

Продовження таблиці 2.1

ORC 110xx (Object Recognizer Chip)	Synaptics	-	-	-	
Pram-256 Chip	UCLi Ltd.	8 (одного нейрона)	-	256 нейронов	33МГц.
SAND	Datafactory	16	-	4	200 MCPS
ACC		16	-	-	
Геркулес	Россия	16	1 Мбайт	64	
Neuro Classifier	Университет Твента, DESY	70 вх. нейронов	-	6 (внутр) 1 вх., 1 вых.	2 x 1010 перекл./с
ANNA	AT&T	Число нейроннов 16-256	4096 весов	-	Число входов у нейрона 256-16.
WSC (Wafer Scale Integration)	Hitachi	-	64 связи на нейрон	576 нейронов	
SASLM2	Mitsubishi	2 (одного нейрона)	-	4096(64x64) нейронов	50 МГц
TOTEM	Kent (Univer UK), di Trento (Italy)	16 (одного нейрона)	-	64 нейрона	30 МГц
Neuron 3120, Neurom 3150	Echelon (США)	8 бит (шина данных)	-	-	Наличие параллельных, последовательных и коммуникацинных портов

Розглянувши основні характеристики елементної бази найпоширеніших нейрочипів можна виділити найпотужніші та найперспективніші (табл. 2.2).

Таблиця 2.2 - Найпотужніші нейрообчислювачі

Назва	Конфігурація	CPS	CPSPW	CPPS	CUPS
NLX420	32-16, 8 bit mode	10M	20K	640M	-
100 NAP	4 chips, 2M wts, 16 bit mantissa	250M	125	256G	64M
WSI (Hitachi)	576 neuron Hopfield	138M	3.7	10G	-
N64000 (Inova)	64-64-1, 8 bit mode	871M	128K	56G	220M
MA16	1 chip, 25MHz	400M	15M	103G	-
ZISC036	64 8 bit element inp. Vector	-	-	-	-
MT19003	4-4-1-, 32 MHz	32M	32M	6.8G	-
MD1220	8-8	9M	1M	142M	-
NI 1000	256 5 bit element inp. Vector	40 000 vec in sec.	-	-	-
L-neuro-1	1-chip, 8 bit mode	26M	26K	1.6G	32M
NM6403	8 bit mode, 50MHz	1200M	150M	77G	-

Отже, серед найпотужніших нейрочипів найбільшу цікавість являє собою нейрочіп NeuroMatrix NM6403, російської компанії Модуль. Основою NeuroMatrix NM6403 є процесорне ядро NeuroMatrixCore (NMC), що являє собою модель високопродуктивного DSP процесора з архітектурою VLI/SIMD. Ядро складається із двох базових блоків: 32-бітного RISC процесора й 64-бітного векторного процесора, що забезпечує виконання векторних операцій над даними змінної розрядності. Є два ідентичних програмувальних інтерфейси для роботи із зовнішньою пам'яттю різного типу й два комунікаційних порти, апаратно сумісних з портами ЦПС TMS320C4x, для можливості побудови багатопроцесорних систем.

2.2.2 Архітектура NeuroMatrix NM6403

Базовими для даного нейропроцесора є обчислення виду:

$$Zi=f(Yi) = f\ (Ui+e(XjWij),\ (i=1,\ ..\ ,\ M;j=,\ ..,N), \qquad (2.2)$$

де Zi - вихідний сигнал і-го нейрона, Xj - j- й вхідний сигнал шару, Ui - зсув і-го нейрона, Wij - ваговий коефіцієнт j- го входу і-го нейрона, Yi - сума зважених входів і-го нейрона, f- функція активації, N - кількість вхідних сигналів шару, M - кількість нейронів у шарі; Операнди Zi, Xi, Ui й Wij представлені в додатковому паралельному коді й можуть мати довільну розрядність. [12]

До основних особливостей нейропроцесора можна віднести:[12]

- можливість роботи із вхідними сигналами (синапсами) і вагою змінної розрядності (від 1 до 64 біт), що задається програмно, що забезпечує унікальну здатність нейропроцесора збільшувати продуктивність зі зменшенням розрядності операндів;
- швидке додавання нових ваг на тлі обчислень;
- (24 операції множення з нагромадженням за один такт при довжині операндів 8 біт);
- V апаратна підтримка емуляції нейромереж великої розмірності;
- реалізація функції активації у вигляді граничної функції або функції обмеження;
- наявність двох широких шин (по 64 розряди) для роботи із зовнішньою пам'яттю будь-якого типу: до 4Мб SRAM і до 16 Гб DRAM;
- наявність двох байтових комунікаційних портів уведення/виведення, апаратно сумісних з комунікаційними портами TMS320C4x для реалізації паралельних розподілених обчислювальних систем великої продуктивності;

- можливість працювати з даними змінної розрядності по різних алгоритмах, реалізованим за допомогою програм котрі зберігаються в зовнішньому ОЗУ.

Основні характеристики процесора NeuroMatrix NM6403[12]

- тактова частота - 50 MHz (20нс - час виконання будь-якої інструкції);
- технологія КМОП 0.5 напівтемних;
- корпус 256BGA;
- напруга харчування від 2.7 до 3.6 В;
- споживана потужність при 50 MHz близько 1.3 Вт;
- умови експлуатації: -60...+85 С.

RiSC-ядро

- 5-ти східчастий 32- розрядний конвеєр;
- 32- і 64- розрядні команди (звичайно виконується дві операції в одній команді);
- два адресних генератори, адресне простір - 16 GB;
- два 64- розрядних програмувальних інтерфейси з SRAM/ DRAM- Поділюваною пам'яттю;
- формат даних - 32- розрядні цілі;
- регістри:
- 8 32- розрядних регістрів загального призначення;
- 8 32- розрядних адресних регістрів;
- спеціальні регістри керування й стану;
- два високошвидкісних комунікаційних порти уведення/висновку,
- апаратно сумісних з портами TMS320C4x.

VECTOR-Співпроцесор

- змінна 64- розрядна довжина векторних операндів і результатів;
- формат даних - цілі числа, упаковані в 64- розрядні блоки, у формі слів змінної довжини від 1 до 64 розрядів кожне;
- підтримка векторно-матричних і матрично-матричних операцій;
- два типи функцій насичення на кристалі;
- три внутрішніх 32х 64-розрядних RAM- Блоки.

Продуктивність:

- скалярні операції:
- 50 MIPS;
- 200 MOPS для 32- розрядних даних;
- векторні операції:
- від 50 до 50.000+ MMAC (мільйонів множень із накопиченням у секунду);
- I/O і інтерфейси з пам'яттю:

– пропускна здатність двох 64- розрядних інтерфейсів з пам'яттю - до 800 Мбайт/с;
– I/O комунікаційні порти - до 20 Мбайт/с кожний.

Технічні характеристики процесора NeuroMatrix NM6403[12]

– число вентилів на кристалі - 100.000;
– розмір кристала - 10 мм * 10.5 мм при технології 0.7 напівтемних;
– споживана потужність - не більше 3 Вт;

пікова продуктивність для байтових операндів - 720 MCPS (мільйонів з'єднань або множень із нагромадженням у секунду) при тактовій частоті 30 Мгц; при бінарних операціях - 8640 MCPS.

2.3 Проектування обчислювальної системи на основі нейронної мережі радіально-базисних функцій

Нейрокомп’ютери з паралельною обробкою даних являють собою багатопроцесорні системи на базі каскадного з’єднання сигнальних процесорів з можливістю паралельної обробки. У структурі такого роду комп’ютерів можна виділити дві основні частини:

– управляючу ЕОМ, що реалізовується на звичайній обчислювальній системі, на базі CISC або RISC процесорів;
– зовнішній апаратний пристрій, котрий під’єднується до головної ЕОМ, і в свою чергу виконує всі основні операції (дане твердження правильне, у разі коли нейрокомп’ютер являє собою плату розширення).

2.3.1 Огляд існуючого рішення нейрокомп’ютера на базі процесора NeuroMatrix 6403

Інструментальний модуль МС4.31 є найбільш поширеним та найпотужнішим (рис. 2.2). Даний модуль призначений для роботи в складі ПЕОМ із системною шиною PCI для відпрацьовування функціонального програмного забезпечення обчислювальних систем на базі процесора Л1879ВМ1.

Модуль містить один Л1879ВМ1 із двома банками однотактової статичної пам'яті по 2 Мбайти (по одному банку на кожній шині процесора). Один банк пам'яті доступний для запису й читання як з боку процесора, так і з боку шини PCI.

На зовнішні рознімання модуля виведені два комунікаційних порти процесора, які призначені для об'єднання декількох модулів або приєднання пристроїв уведення/висновку. Комунікаційні порти можуть бути використані в

якості отладочного інтерфейсу для сполучення ПЕОМ із бортовою апаратурою на базі процесора Л1879ВМ1.

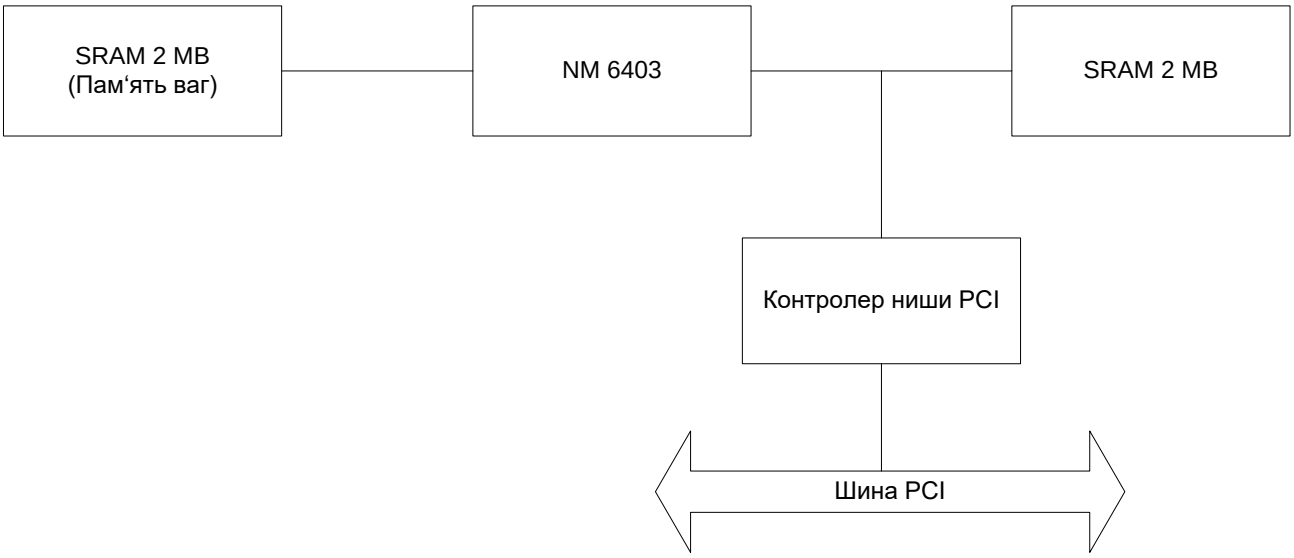


Рисунок 2.2 - Структурна схема модулю МС4.31

З боку шини PCI модуль видний як 32- х розрядний провідний пристрій у просторі адрес висновку.

2.3.2 Розробка нейрокомп’ютера з паралельною обробкою даних

Отже, проведемо вдосконалення даного модулю. Для паралельної обробки даних використаємо N+1 процесор, де 1 процесор є головним, який керує іншими процесорами. Дана структура є кластером з процесорів. Вдосконалена схема нейрокомп’ютера представлена на рис.2.3.

В даній схемі, представлено один головний процесор, головними функціями котрого є здійснення загального управління, та до 4 процесорів котрі будуть виконувати разом з головним процесором паралельні обчислення. Усі процесори, разом утворюють кластер процесорів з загальною шиною та даними. Також використовується 4 МВ пам’яті для даних, розподіленої на два модулі, та 2 МВ пам’яті для пам’яті ваг.

Головною особливістю даного нейрокомп’ютера (див. рис.2.3) є те, що є наявність розширення кількості процесорів (до 4 штук), що дозволяє підлаштовувати нейрокомп’ютер під свої потреби. Також, можлива реалізація незалежних задач на процесорах, що виконуються незалежно одне від одного, але швидкість виконання не зросте, адже процесори використовують загальну пам’ять ваг.

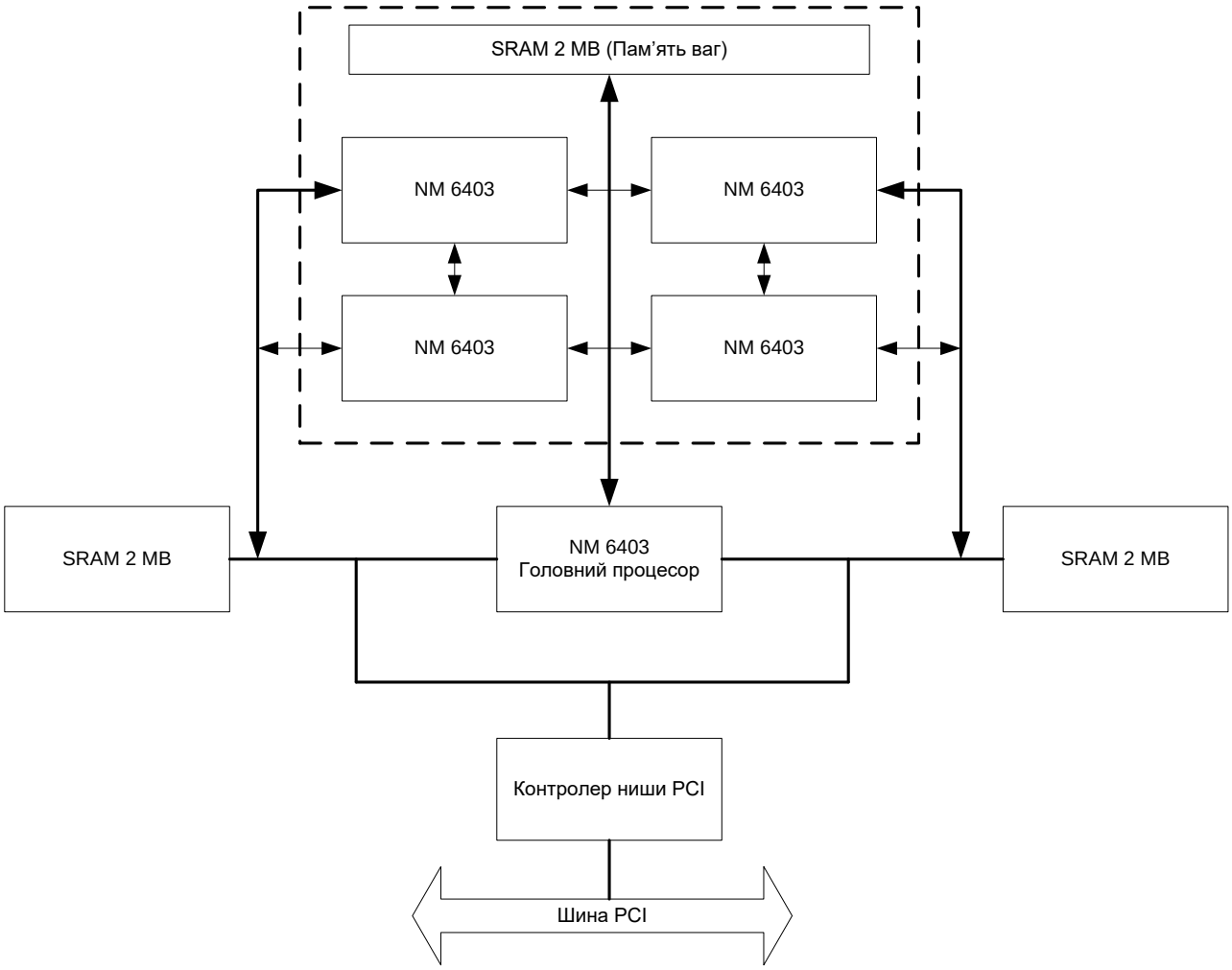


Рисунок 2.3 - Структурна схема модулю нейрокомп'ютера для нейромережі РБФ №1

Так, як розробка нової розширеної плати для 4 та більше процесорів, може виявитися досить дорогим задоволенням, то збільшення кількості процесорів та організацію паралельної обробки можливо досягнути завдяки збільшенню кількості модулів МС4.31 або їм подібних, та розміщення їх у системних шинах розширення PCI. Схема розміщення представлена на рис.2.4

В даній схемі представлена схема нейрокомп'ютера, у котрому кількість нейропроцесорів може змінюватися від 1 до N, де N – кількість PCI вільних шин. Дана схема являє собою кластер, в котрій головним вузлом виступає процесор ЕОМ, результати розрахунків зберігаються у оперативній пам'яті комп'ютера. Головний процесор (вузол) розподіляє задачі між вузлами кластеру. Можливе створення доступу до даного комп'ютера через мережу Ethernet, або подібні.

До переваг даної схеми можна віднести:

– можливість розширення кількості процесорів до кількості PCI шин комп'ютера;
– реалізація незалежної обробки даних декількома процесорами;

– паралельність виконання задач;

До недоліків даної схеми можна віднести:

– зайняття шин PCI;
– використання зайвої електричної енергії;

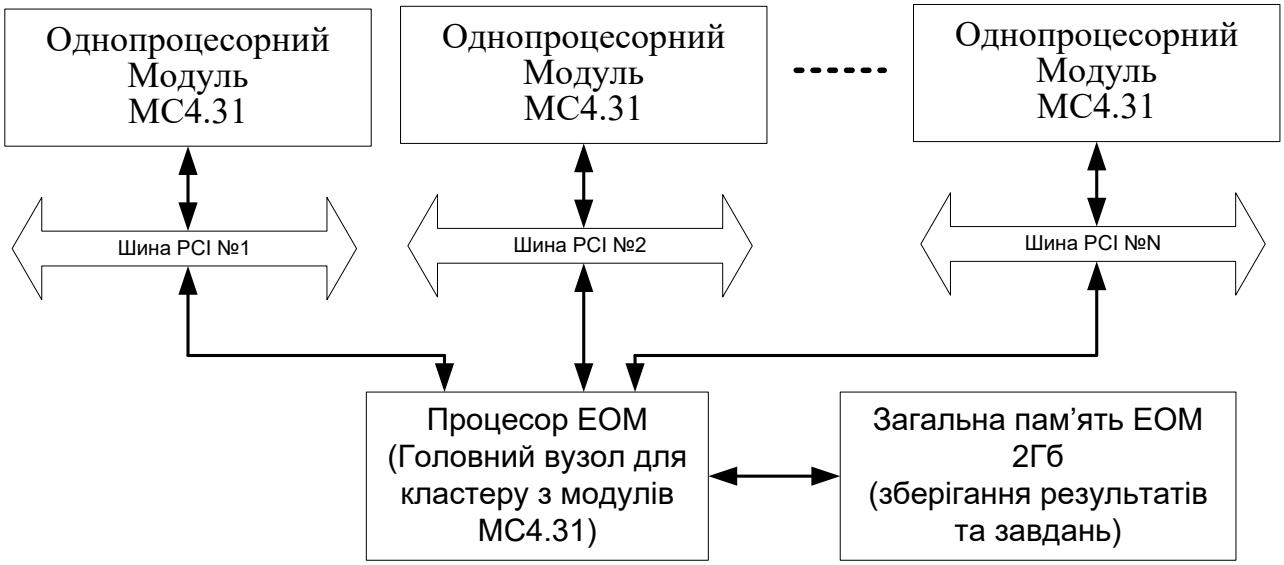


Рисунок 2.4 - Схема нейрокомп'ютера на основі нейромережі РБФ №2

Іншим варіантом, розроблення нейрокомп'ютера для паралельної обробки є розміщення модулів МС4.31 не в одній ЕОМ, а в декількох. Схема даного нейрокомп'ютера представлена на рис. 2.5

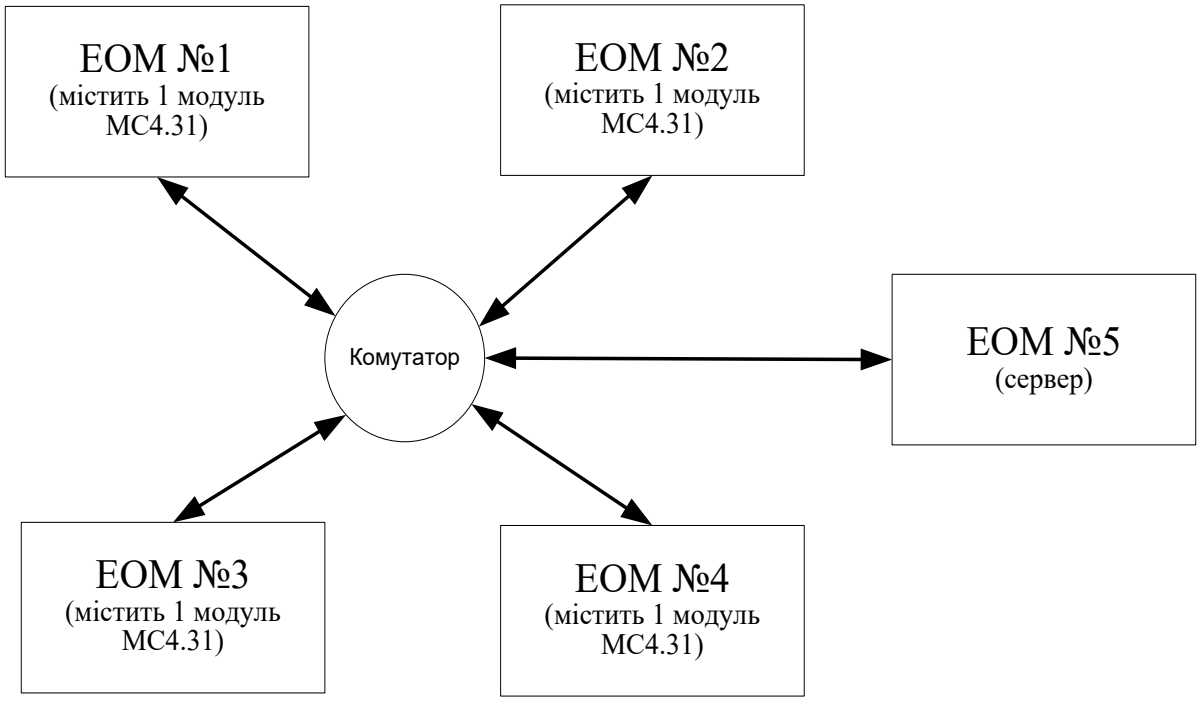


Рисунок 2.5 - Схема нейрокомп'ютера на основі нейромережі РБФ №3

Представлена вище схема являє собою кластер з комп’ютерів, де головним вузлом може бути як ЕОМ котра містить модуль МС4.31 так і окремий сервер. Даний кластер являє собою деяку мережу, з топологією зірка. За допомогою ЕОМ сервера можна організувати на базі університету, або навіть міста доступ до даного нейронного комп’ютера – внутрішньо мережний
доступ, або доступ через мережу Internet.

До переваг даної схеми можна віднести:

– незалежність кожного модулю оди від одного;
– паралельність обчислень;
– незалежне виконання задач;
– організація мережевого доступу до нейрокомп’ютера;

До недоліків даної схеми можна віднести:

– залежність роботи «нейронного кластера» від комутатора;
– використання додаткової електроенергії;

Якщо, ж ресурсів нейрокомп’ютера не вистачає, або у випадку створення над продуктивного нейрокомп’ютера можлива наступна схема нейрокомп’ютера для реалізації паралельної обробки, представлена на рис.2.6.

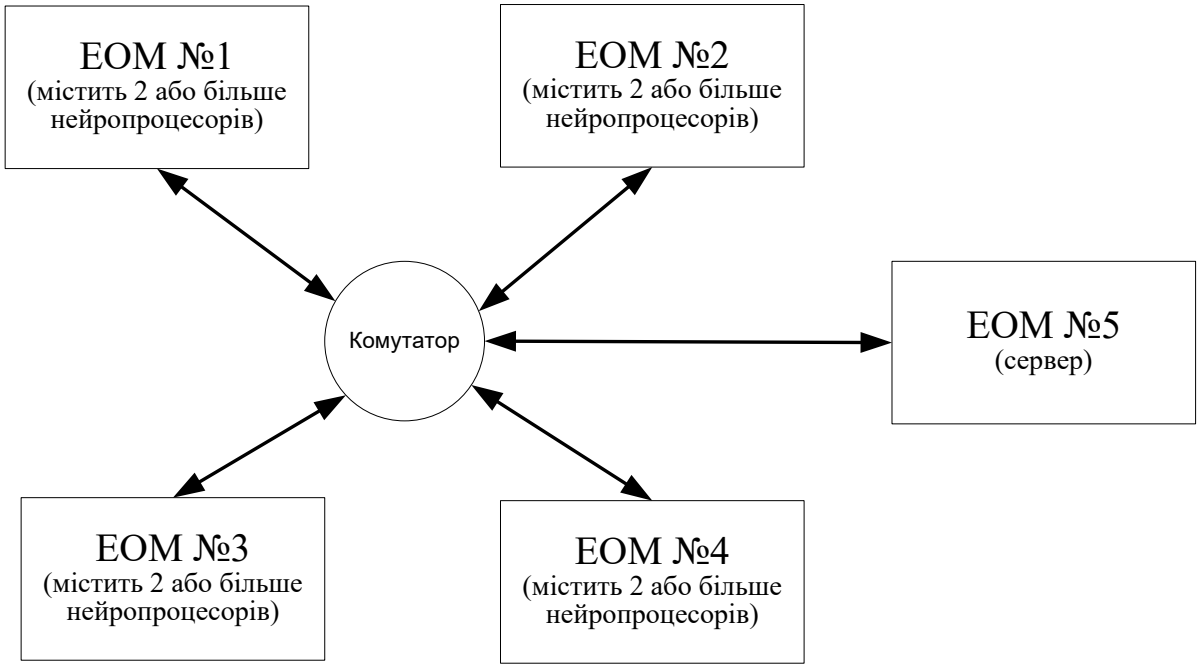


Рисунок 2.6 Схема нейроком’ютера на основі нейромережі РБФ №4

До особливостей даної схеми можна віднести, наявність у кожному вузлі кластеру багатопроцесорного модулю, за рахунок котрого збільшується продуктивність даної системи.

До переваг даної схеми можна віднести:

– незалежність кожного модулю оди від одного;

– паралельність обчислень;
– незалежне виконання задач;
– організація мережевого доступу до нейрокомп’ютера;
– збільшення продуктивності даної системи в декілька разів;

До недоліків даної схеми можна віднести:

– залежність роботи «нейронного кластера» від комутатора;
– використання додаткової електроенергії;
– складність реалізації програмного забезпечення для даної системи;
– головним вузлом даної системи виступає сервер.

Висновки до розділу:

1. Серед найбільш потужних нейрочипів та найбільш раціональним з категорії ціна/якість є процесор NeuroMatrix NM6403;

2. Розробка обчислювальної системи (нейрокомп’ютера) на основі нейромережі РБФ відбувалась на базі модулю МС4.31, фірми «Модуль»;

3.У приведених схемах (див. рис.2.4-2.6) головною особливістю є те, що в основі кожної системи є кластер;

3. Найбільш продуктивнішим нейрокомп’ютером є схема представлена на рис.2.6, котра характеризується незалежністю виконання задач, збільшенням продуктивності обчислень, та можливістю мережевого доступу до «нейронного кластеру».

РОЗДІЛ 3

АНАЛІТИЧНА ЧАСТИНА

3.1 Аналіз нейромереж типу РБФ

РБФ нейрони навчаються шляхом підбору центру і відхилення кожної з них. Для класифікатора в якості центру вибирається центр кластера в просторі ознак, компактно що містить образи одного і того ж класу. У найпростішому випадку, якщо клас задається одним ідеальним чином, цей образ і буде вектором t - центром РБФ осередки. Параметр розкиду кожного нейрону вибирається залежно від величини радіуса кластера або відстані до сусідніх центрів. Ряд авторів рекомендує вибирати як половину відстані до найближчого центру осередки, відповідної іншого класу. Кількість РБФ осередків вибирається таким чином, щоб покрити гауссовими дзвонами всі класи (див. рис. .3.1).

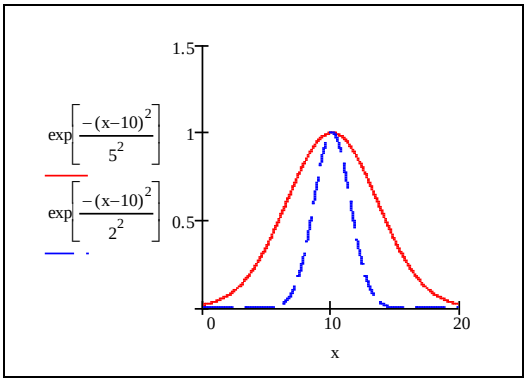


Рисунок 3.1 - Приклади функцій РБФ нейронів з однаковим центром і різним розкидом

Ця функція має максимум, рівний 1, коли вхід дорівнює 0. Коли відстань між векторами w і p зменшується, вихід радіальної базисної функції збільшується. Таким чином, радіальний базисний нейрон діє як індикатор, який формує значення 1, коли вхід p ідентичний вектору ваг w. Зсув b дозволяє коригувати чутливість нейрона radbas. Наприклад, якщо нейрон мав зсув 0.1, то його виходом буде 0.5 для будь-якого вектора входу p і вектора ваги w при відстані між векторами, що дорівнює 8.333, або 0.833 / b.

Радіальна базисна мережа складається з двох шарів: прихованого радіального базисного шару, що має S1 нейронів, і вхідного лінійного шару, що має S2 нейронів (рис. 2.2).

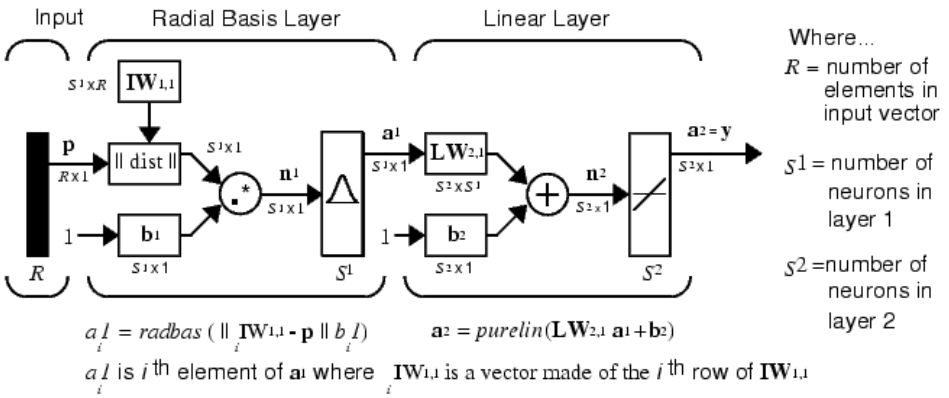


Рисунок 3.2 – Архітектура нейромережі типу РБФ

Входами блоку || dist || на цьому малюнку є вектор входу p і матриця ваг IW1,1, а виходом - вектор, відбулась з S1 елементів, які визначаються відстанями між i-м вектором входу p і i-й вектор-рядком IW1,1 матриці ваг. Таку вектор-рядок будемо називати вектором ваг i-го нейрона. Вихід блоку || dist || множиться поелементно на вектор зміщення b1 і формується вхід функції активації.

Вихідний шар РБФ мережі зазвичай складається з підсумкових нейронів:

$$y_k = \sum_{j=1}^{h} w_{jk} g_j \qquad (3.1)$$

3.2 Порівняння мереж РБФ та багатошарових перспетронів

Мережі на основі радіальних базисних функцій і багатошаровий персептрон (MLP) є прикладами нелінійних багатошарових мереж прямого поширення. І ті й інші є універсальними аппроксіматорамі. Таким чином, не дивно, що завжди існує мережа РБФ, здатна імітувати багатошаровий персептрон (і навпаки). Однак ці два типи мереж відрізняються за деякими важливими аспектами.

1. Мережі RBF (в своїй основній формі) мають один прихований шар, в той час як багатошаровий персептрон може мати більшу кількість прихованих шарів.

2. Зазвичай обчислювальні (computational) вузли багатошарового персептрона, розташовані в прихованих і вихідному шарах, використовують одну і ту ж модель нейрона. З іншого боку, обчислювальні вузли прихованого шару мережі РБФ можуть докорінно відрізнятися від вузлів вихідного шару і служити різним цілям.

3. Прихований шар в мережах РБФ є нелінійним, в той час як вихідний - лінійним. У той же час приховані і вихідний шари багатошарового персептрона, використовуваного в якості класифікатора, є нелінійними. Якщо багатошаровий персептрон використовується для вирішення задач нелінійної регресії, як вузли вихідного шару зазвичай вибираються лінійні нейрони.

4. Аргумент функції активації кожного прихованого вузла мережі РБФ є Евклидову норму (відстань) між вхідним вектором і центром радіальної функції. У той же час аргумент функції активації кожного прихованого вузла багатошарового персептрона - це скалярний добуток вхідного вектора і вектора синаптичних ваг даного нейрона.

5. Багатошаровий персептрон забезпечує глобальну апроксимацію нелінійного відображення. З іншого боку, мережу РБФ за допомогою

експоненціально зменшуються локалізованих нелінейностей (тобто функцій Гаусса) створює локальну апроксимацію нелінійного відображення.

Це, в свою чергу, означає, що для апроксимації нелінійного відображення за допомогою багатошарового персептрона може знадобитися менше число параметрів, ніж для мережі RBF при однаковій точності обчислень.

Лінійні характеристики вихідного шару мережі RBF означають, що така мережа тісніше пов'язана з персептроном Розенблатта, ніж з багатошаровим персептроном. Проте, мережі RBF відрізняються від цього персептрона тим, що здатні виконувати нелінійні перетворення вхідного простору. Це було добре продемонстровано на прикладі рішення задачі XOR, яка не може бути вирішена жодним лінійним персептроном, але з легкістю вирішується мережею RBF.

3.3 Використання радіальної базисної мережі для вирішення задачі апроксимації функції

Нехай задана функція f (x) графічно без аналітичного опису (рис. 3.3).

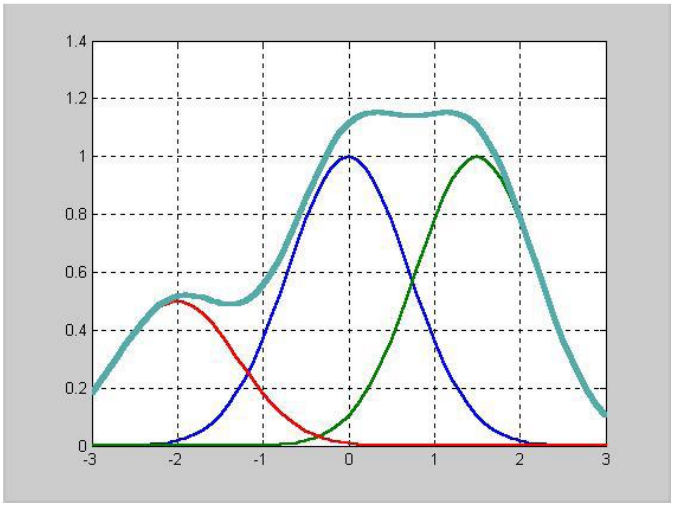


Рисунок 3.3 – Приклад деякої функції

Необхідно підібрати таку аналітичну залежність, яка із заданою точністю описувала б цю функціональну залежність. В даному випадку використовується сума радіальних базисних функцій:

$$f(x) = \sum_{i=1}^{N} \alpha_i \varphi_i(x) \tag{3.2}$$

Де φ(x) – РБФ. Ідея апроксимації може бути представлена графічно наступним чином та на сонові використання пакету MatLab. Розглянемо зважену суму трьох радіальних базисних функцій, заданих на інтервалі [-3; 3].

```
clear, p=-3:.1:3;
a1=radbas(p);
a2=radbas(p-1.5);
a3=radbas(p+2);
```

```
a=a1+a2*1+a3*0.5;
figure(1),clf,
plot(p,a,p,a2,p,a3*0.5,'LineWidth',1.5),hold on,
plot(p,a,'LineWidth',3,'Color',[1/2,2/3,2/3]),grid on
legend('a1','a2','a3*0.5','a')
```

Як випливає з рис. 3.2, при завданні вектора входу кожен нейрон радіального базисного шару видасть значення відповідно до того, як близький вектор входу до вектору ваг кожного нейрона.

Радіальні базисні нейрони з векторами ваг, що значно відрізняються від вектора входу p, матимуть виходи, близькі до 0, і їх вплив на виходи лінійних нейронів буде незначно.

Навпаки, радіальний базисний нейрон з вектором ваг, близьким до вектору входу p, видасть значення, близьке до 1, і це значення буде передано на лінійний нейрон з вагою, відповідним вихідному прошарку.

Отже, якщо тільки один радіальний базисний нейрон має вихід 1, а всі інші мають виходи, рівні або дуже близькі до 0, то вихід лінійного шару буде дорівнює вазі активного вихідного нейрона.

Однак це винятковий випадок. Зазвичай вихід формують кілька нейронів з різними значеннями ваг.

Згідно рис. 3.1, виконаємо ручної розрахунок для аналізу a1, a2, a3.

$$radbas(n) = e^{-n^2}$$

$$p = 2; w_1 = -2; w_2 = 0; w_3 = 1,5; b_1 = b_2 = b_3 = 1,$$

$$\| p - w_1 \| b_1 = | 2 - (-2) | * 1 = 4,$$

$$\| p - w_2 \| b_2 = | 2 - 0 | * 1 = 2,$$

$$\| p - w_3 \| b_3 = | 2 - 1,5 | * 1 = 0,5,$$

$$radbas(n_1) = e^{-4^2} - e^{-16} \approx 0,$$

$$radbas(n_2) = e^{-22} = e^{-4} \approx 0,000001,$$

$$radbas(n3) = e^{-0.52} = e^{-0,25} \approx 0,8$$

Як видно з рис. 3.2, в точці p = 2 підсумовується практично тільки вихід третього нейрона. Вплив в цій точці першого і другого нейронів незначно.
Виходи a1 і a2 близькі до 0. Вихід a3 = 0,8 близький до 1.

Аналізуючи рис. 3.2, приходимо також до висновку, що розкладання по радіальних базисних функціях забезпечує необхідну гладкість. Тому їх застосування для апроксимації довільних нелінійних залежностей цілком виправдано.

Розглянемо приклад формування радіальної базисної мережі в системі MATLAB для вирішення завдання апроксимації. Сформуємо навчальну множину і поставимо допустиме значення функціоналу помилки, рівне 0.01, параметр впливу визначимо рівним 1 і будемо використовувати итерационную процедуру формування радіальної базисної мережі:

```
P = -1:.1:1;
T = [-.9602 -.8770 -.0729 .3771 .6405 .6600 .4609 .1336 ...
      -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 ...
     .3449 .1816 -.0312 -.2189 -.3201];
GOAL = 0.01;
SPREAD = 1;
net = newrb(P, T,GOAL, SPREAD); % Создание сети
net.layers{1}.size %Число нейронов в скрытом слое
NEWRB, neurons = 0, SSE = 3.69051
ans =
6
```

Для заданих параметрів нейронна мережа складається з 6 нейронів і забезпечує наступні можливості апроксимації нелінійних залежностей після навчання. Моделюючи сформовану нейронну мережу, побудуємо Апроксимаційні криву на інтервалі [-1; 1] з кроком 0.01 для нелінійної залежності.

```
figure(1), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
X = -1:.01:1;
    Y = sim(net, X); %Моделирование сети
```

З аналізу рис. 3.4 випливає, що при невеликій кількості нейронів прихованого шару радіальна базисна мережа досить добре апроксимує нелінійну залежність, задану навчальним безліччю з 21 точки.

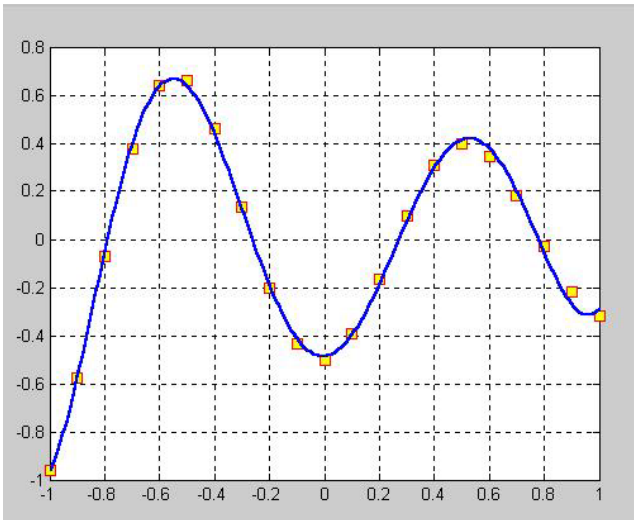


Рисунок 3.4 – Результат апроксимації

У демонстраційних прикладах досліджується вплив параметра SPREAD на структуру радіальної базисної мережі і якість апроксимації. У демонстраційному прикладі demorb3 параметр впливу SPREAD встановлений рівним 0.01. Це означає, що діапазон перекриття вхідних значень становить лише ± 0.01, а оскільки навчальні входи задані з інтервалом 0.1, то вхідні значення функціями активації не перекриваються. Це призводить до того, що, по-перше, збільшується

кількість нейронів прихованого шару з 6 до 13, а по-друге, не забезпечується необхідною гладкості функції, що апроксимується (рис. 3.5):

```
>> P = -1:.1:1;
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -.3930 -.1647 .0988
.3072 .3960 .3449 .1816 -.0312 -.2189 -.3201];
GOAL = 0.01;
SPREAD = 0.01;
net = newrb(P,T,GOAL,SPREAD);
net.layers{1}.size % Число нейронов в скрытом слое
NEWRB, neurons = 0, MSE = 0.176192
ans =
   13
```

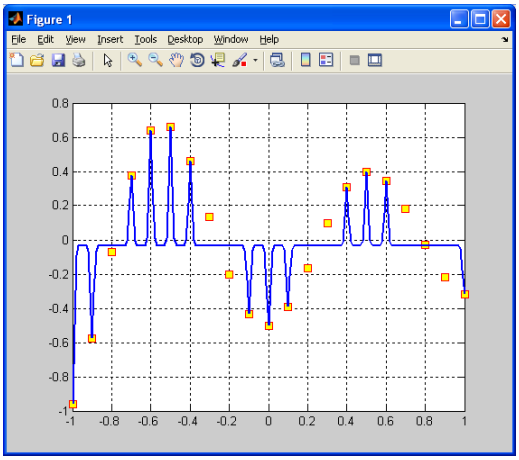


Рисунок 3.5 – Результат моделювання

```
figure(1), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
X = -1:.01:1;

Y = sim(net,X); % Моделирование сети
    plot(X,Y,'LineWidth',2), grid on
```

В цьому випадку всі функції активації перекриваються і кожен базовий нейрон видає значення близьке до 1 для всіх значень входів. Це призводить до того, що мережа не реагує на вхідні значення. Функція newrb намагатиметься будувати мережу, але не зможе забезпечити необхідну точність:

```
GOAL = 0.01; SPREAD = 12;
net = newrb(P,T,GOAL,SPREAD);
net.layers{1}.size
NEWRB, neurons = 0, MSE = 0.176192
ans =
```

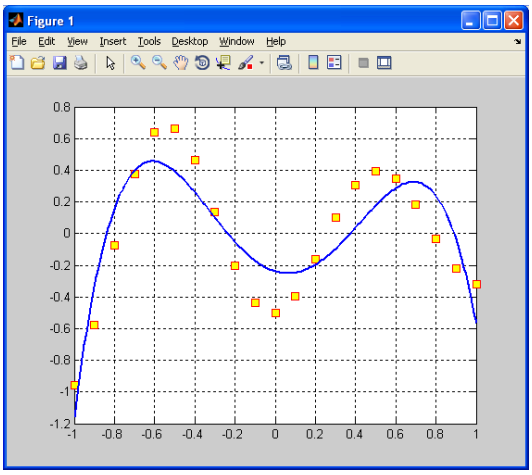


Рисунок 3.6 – Результат моделювання

Висновки до розділу:

В результаті виконання даного розділу було виконано аналіз роботи нейромережі радіально-базисних функції. Встановлено її основні переваги та недоліки у порівнянні із багатошаровим персептроном.

4 ДОСЛІДНА ЧАСТИНА

Для вирішення диференційних рівнянь в часткових похідних ефективніше всього використовувати нейронні мережі. В сучасний час великий інтерес викликають методи вирішення з використанням радіально-базисної функції. Ці методи можуть бут ефективно реалізовані на радіально-базисних нейронних мережах.

Ідея методу вирішення на РБФ-мережах – апроксимація невідомого рішення за допомогою функцій спеціального виду, аргументами яких є відстань. Найбільш часто використані на практиці радіальні функції гаусівського типу за своїм типом мають локальний характер і приймають не нульові значення лише довкола певного центра. Це дозволяє легко встановити залежність між параметрами базисних функцій і фізичним розміщенням даних в багатомірному просторі.

У порівнянні з багатошаровими мережами, що мають сігмоідальні функції активування, РБ нейронні мережі відрізняються деякими специфічними властивостями, що в свою чергу забезпечують більш просте відображення характеристик змодельованого процесу.

Метою є дослідження особливостей навчання РБ нейронних мереж для вирішення рівнянь математичної фізики.

У процесі навчання встановлюються усі параметри мережі: ваги, центри та ширина. При цьому будемо використовувати два методи навчання нейронної мережі: безсітковий метод, що допускає розміщення нейронів і контрольних точок в довільних точках області вирішення та поза нею (лише для нейронів), а також метод заснований на використанні кінцево-різничній апроксимації рівняння, коли контрольні точки розміщуються на регулярній сітці, а центри нейронів у довільних точках.

Основну увагу приділимо дослідженню наступних питань:

-дослідження фіксованого і випадкового характеру розміщення контрольних точок у яких проводиться перевірка наближення вирішення на задовільненя обчислюємого рівняння (вираховується функціонал властивостей)

-дослідження впливу коефіцієнтів швидкості навчання на сам процес навчання

-знаходження формули, обчислюємого у процесі навчання коефіцієнта швидкості навчання найбільш значимих параметрів мережі

-ваг

-дослідження впливу початкового розміщення центрів на процес навчання

4.1 Вирішення диференційних рівнянь в часткових похідних на РБ нейронних мережах

Радіально-базисна нейронна мережа (рис.1) представляє собою мережу з одним прихованим шаром.

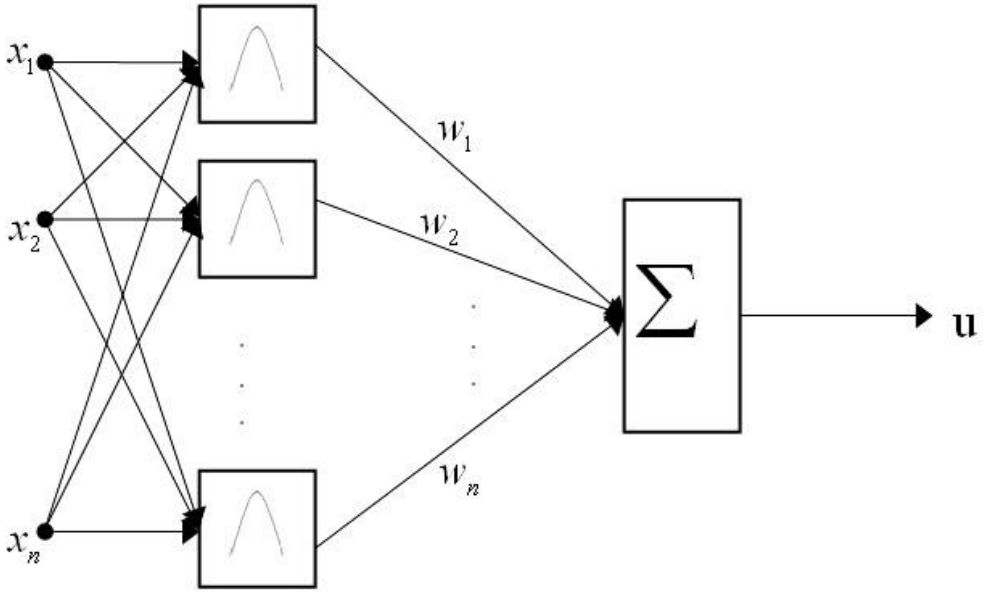


Рис. 1 Радіально-базисна нейронна мережа

Прихований шар перетворює вхідний вектор **X** із застосуванням радіально-базисної функції (RBF). Практично використовуються різні радіально-базисні функції. У подальшому будемо використовувати найбільш часто використовувану функцію – Гаусіан, що має наступний вид для *k*–го нейрона:

$$\phi_k(\mathbf{X}) = \exp(-r_k^2 / a_k^2), \qquad (1)$$

де **X** – вхідний вектор; r_k – радіус,

$$r_k = \|\mathbf{X} - \mathbf{C}_k\|;$$

$\mathbf{C}_k$ – вектор центру RBF; a – параметр функції, що зветься шириною.

Вихідний шар мережі представляє собою лінійний суматор, а вихід мережі описується рівнянням:

$$u = \sum_{k=1}^{N} w_k \varphi_k(\mathbf{X}), \qquad (2)$$

Де w_k – вага, зв'язуюча вихідний нейрон с k-м нейроном прихованного шару.

Вираз (2) представляє собою формулу методу колокації для апроксимації функцію та рішення диференційних рівнянь. Розглянемо метод колокації, реалізованого за допомогою РБФ, на прикладі вирішення лінійного двумірного еліптичного диференційного рівняння в часткових похідних.
Нехай $\Omega \subset \mathbf{R}^d$ – d -мірная область і $\partial\Omega$ – межа цієї області.
В операторній формі задача запишется у вигляді:

$$Lu(\,)\mathbf{X} = f(\,)\mathbf{X}\,, \quad \mathbf{X}\in\Omega\,, \qquad (3)$$

З межевими умовами

$$Bu(\,)\mathbf{X} = g(\mathbf{X}), \quad \mathbf{X}\in\partial\Omega\,, \qquad (4)$$

де L – лінійний диференційний оператор(наприклад, лапласіан); B – оператор межевих умов.

Задамо множину точок колокації $\mathbf{X}$ $\mathbf{x}$={ $_1$,..., $\mathbf{x}_N$}$\subset \Omega$, в яких буде знаходитися вирішення задачі (3)-(4), частину точок розмістимо в середині області Ω (підмножини $\mathbf{X}_\Omega$), а частину на межі області $\partial\Omega$ (підмножини $\mathbf{X}_{\partial\Omega}$).

Оберемо центри і ширину РБФ (1), за допомогою яких буде виконуватися апроксимація рішень. Центри в загальних випадках можуть можуть не співпадати з точками колокації. Підставляючу вираз (1) в (3) та (4) отримаємо систему лінійних алгебраїчних рівнянь.

$$\mathbf{AW} = \mathbf{F}$$

Де $\mathbf{A} = \begin{bmatrix} \mathbf{\Phi} \\ \mathbf{L} \end{bmatrix}$ - блочна колокаційна матриця, елементи якої дорівнюють:

$$\mathbf{\Phi}_{ij} = \varphi\left(\left\|\mathbf{X}_i - \mathbf{C}_j\right\|\right), \qquad \mathbf{X}_i \in \mathbf{X}_{\partial\Omega},$$
$$L_{ij} = \mathbf{L}\varphi\left(\left\|\mathbf{X}_i - \mathbf{C}_j\right\|\right), \qquad \mathbf{X}_i \in \mathbf{X}_{\Omega},$$

Причому, центри C_j можуть розміщуватися і поза областю Ω;
$\mathbf{W}$ =[w $w_{1\,2}$, ,..., w_n]$^{\mathrm{T}}$ – вектор ваг; $\mathbf{F}$ – вектор, створений х компонентів вектору правої частини рівняння (3) та (4).

Систем (5) у загальному вигляді має прямокутну матрицю і для її вирішення можна застосувати метод сингулярного розкладу. Такий підхід з вибором центрів, ширини та визначення ваг з рішень системи (5) застосовується у багатьох модифікаціях методу, заснованого на РБФ. Заміщуючи вхідні в Lu

похідні різносними відношеннями, отримаємо різничний вираз $L\,u_{h\,h}$, чим приходимо до методу, заснованого на використанні кінцево-різничної апроксимації рівняння.

Визначивши вирішення в точках колокації, за (2) можна знайти вирішення в довільних точках області вирішення. Точки колокації можна розміщувати довільним чином (в тому числі і випадковим), тобто метод є безсітковий. Знаючі диференційні оператори (3)-(4) та вираз вирішення (2), можна аналітично розраховувати просторові похідні вирішення в довільних точках.

Можливо використання РБ нейронних мереж для вирішення диференційних рівнянь в часткових похідних. При цьому ваги, центри та ширина знаходяться шляхом навчання мережі, що забезпечує більш точне та гнучкіше вирішення.

4.2 **Градієнтний алгоритм навчання мережі**

Розглянемо градієнтний алгоритм навчання РБ нейронної мережі на прикладі модельної задачі «Рівняння Пуассона»:

$$\frac{\partial^2 u}{\partial^2 x}+\frac{\partial^2 u}{\partial^2 y}=\sin(\pi x)\cdot\sin(\pi y),\quad (x,y)\in\Omega, \qquad (6)$$

З межевими умовами Діріхле по всій межі області

$$u=0,\quad (x,y)\in\partial\Omega. \qquad (7)$$

В якості базисної функції використаємо Гаусіан (1), що має вигляд для нейрона k

$$\varphi_k(x,y)=e^{-r_k^2/a_k^2}, \qquad (8)$$

де $r_k=\sqrt{(x-x_{ck})2+(y-y_{ck})^2}$ – відстань від центру (x_{ck}, y_{ck}) нейрону k до точки (x,y), в якій проходить пошук рішення; a_k – ширина нейрону.

Апроксимація рішення має вигляд:

$$u(x,y)=\sum_{k=1}^{m} w_k\varphi_k, \qquad (9)$$

де w_k – вага; m – кількість нейронів.

Навчання мережі зводиться до налаштування ваг, розміщення центрів та ширини нейронів, мінімізуючих функціонал якості (функціонал помилки). Функціонал помилки $I(c, a, w)$ визначається при першому і другому підході як сума квадратів нев'язок, що отримані при

підстановці в рівняння або при кінцево-різничну апроксимацію рівняння до внутрішніх та межевих точок.

Як в першому, так і в другому підході для навчання мережі використовується градієнтний алгоритм навчання, одночасно оптимізуючий ваги, центри та ширину. Він може бути побудований у вигляді послідовності двох кроків.:

1) Зафіксувавши центри та ширину, намагаємося знайти ваги, що мінімізують функціонал помилки

$$w_i^{(n)} = w_i^{(n-1)} - \eta^{(n-1)} \frac{\partial I\left(c_i^{(n-1)}, a_i^{(n-1)}, w_i^{(n-1)}\right)}{\partial w_i^{(n-1)}}, \qquad (10)$$

де n – номер циклу навчання; $\eta^{(n-1)}$ – коефіцієнт швидкості навчання.

2) Зафіксувавши $w_i^{(n)}$ знайдемо центри та ширину, мінімізуючий функціонал помилки

$$c_i^{(n)} = c_i^{(n-1)} - \beta^{(n-1)} \frac{\partial I\left(c_i^{(n-1)}, a_i^{(n-1)}, w_i^{(n)}\right)}{\partial c_i^{(n-1)}},$$

$$a_i^{(n)} = a_i^{(n-1)} - \alpha^{(n-1)} \frac{\partial I\left(c_i^{(n)}, a_i^{(n-1)}, w_i^{(n)}\right)}{\partial a_i^{(n-1)}},$$

Де η, β, α – швидкості (коефіцієнти) навчання. Процес уточнення параметрів продовжується до досягнення мінімуму функціоналу.

4.3 Безсітковий метод навчання мережі для вирішення диференційних рівнянь в частково-похідних

Радіально-базисна мережа явним чином апроксимує похідні функції $u(x)$. Похідні функції $u(x)$ розраховуються наступним чином:

$$u_{j\ldots l}(x) = \frac{\partial^k u}{\partial x_j \ldots \partial x_l} = \sum_{i=1}^{m} w^{(i)} \frac{\partial^k \phi^{(i)}}{\partial x_j \ldots \partial x_l}.$$

Функціонал помилки

$$I(w,c,a) = \sum_{i=1}^{N}\left[\frac{\partial^2 u(x_i,y_i)}{\partial x^2} + \frac{\partial^2 u(x_i,y_i)}{\partial y^2} - f(x_i,y_i)\right]^2 + \lambda \sum_{j+1}^{K}\left[u(x_j,y_j) - p_j\right]^2, \qquad (11)$$

Де λ – штрафний множник; p_j – значення межевих умов першого роду в точці j межі; N та K – кількість внутрішніх та межевих контрольних точок.

Представлення вирішення у формі (9) та вид базисних функцій (8) дозволяють обчислити часткові похідні від вирішення:

$$\Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 4\sum_{k=1}^{m}\left[w_k e^{-\frac{r_k^2}{a_k^2}} \frac{r_k^2 - a_k^2}{a_k^4} \right].$$

Функціонал помилки дорівнює

$$I = \sum_{i=1}^{N}\left\{ 4\sum_{k=1}^{m}\left[w_k e^{-\frac{r_{ik}^2}{a_k^2}} \frac{r_{ik}^2 - a_k^2}{a_k^4} \right] - f_i \right\}^2 + \lambda \sum_{j=1}^{K}\left[\sum_{k=1}^{m} w_k e^{-\frac{r_{ik}^2}{a_k^2}} - p_i \right]^2 . \qquad (12)$$

Обчислимо градієнти функціонала за параметрами мережі.

Сучасні наукові дослідження характеризуються, перш за все, впровадженням сучасних математичних методів, а також появою новітніх комп'ютерних технологій, що робить можливим дослідження складних явищ і процесів. На сьогодні, зважаючи на швидкий процес розвитку і впровадження новітніх технологій, задача прогнозування є ще більш актуальною.

Прогнозування – це одна із найнеобхідніших і водночас найскладніших задач інтелектуального аналізу даних. Часто у процесі прогнозування виникають труднощі, пов'язані з недостатньою кількістю й якістю вхідних даних, неоднорідністю середовища, в якому протікає процес, впливом тих чи інших суб'єктивних факторів. Прогнозування, зазвичай, здійснюється з деякою похибкою, оскільки вхідні дані можуть бути неповними і неточними, а від цього й залежить, у свою чергу, модель прогнозу.

Сьогодні існує чимало праць, в яких вирішується задача побудови прогнозу на основі ймовірнісних методів і суб'єктивних знань експертів. Дослідження в даній галузі проводили такі відомі науковці, як Дж.Бокс, В.П.Боровіков, Г.І.Івченко [1, 2] та ін.

Проте вищезгадані методи мають ряд недоліків, зокрема [3, 4]:

-відсутність у моделі уявлень щодо структури й системи зв'язків реального об'єкта;

-трудність побудови моделей за умови, що дані мають зміщення стосовно один одного;

-велика чутливість одержаних результатів до недостатньої інформації (неповних вхідних даних) та їх зашумленості;

-недостатня точність прогнозу;

-залежність результату прогнозу від компетентності аналітика в тій чи іншій предметній області.

Зазначені недоліки породжують необхідність постановки і вирішення задачі, що полягає у використанні нових математичних моделей на основі методів штучного інтелекту, алгоритмів та спеціалізованого програмного забезпечення, які підвищують точність і надійність прогнозу, можуть працювати у випадку недостатньої інформації чи її зашумленості, що дає можливість отримати бажаний результат за короткий час.

Вибір і основа математичної моделі є центральним моментом задачі прогнозування. На практиці нерідко трапляється, що внаслідок тих чи інших причин одержати математичну модель, яка адекватно відбиває властивості досліджуваного об’єкта, надзвичайно складно. Одним із методів вирішення задачі прогнозування є застосування математичних моделей, які засновані на використанні апарата штучних нейронних мереж (ШНМ) з радіально-базисними функціями (РБФ). Даний апарат ШНМ з РБФ має добре розвинену методологію структурного моделювання й методів навчання, які базуються на добре розвиненій теорії програмування.

Дослідженням у сфері ШНМ з РБФ присвячено праці таких відомих науковців, як Є.В.Бодянський, О.Г.Руденко, А.І.Галушкін, В.І.Литвиненко, А.О.Фефелов, О.О.Дідик, Є.Горшков, В.Колодяжний [5-8] та ін.

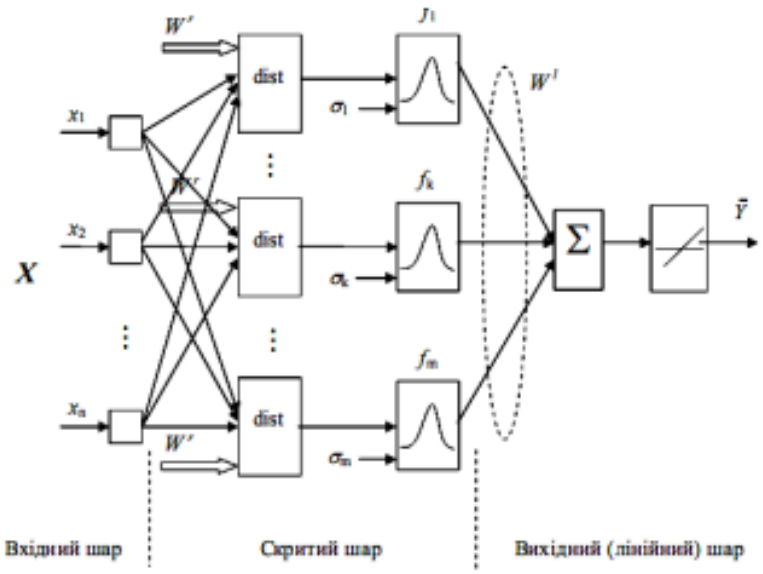
Штучні нейронні мережі з радіально-базисними функціями володіють достатніми прогностичними властивостями, що дає можливість моделювати системи з глибокою нестабільністю, будувати моделі динаміки нестаціонарних об’єктів та прогнозувати випадкові процеси.

Зазвичай прогноз виходить помилковим, та похибка залежить від прогнозуючої системи, яка використовується. Чим більше надано ресурсів для прогнозу, тим кращий і точніший результат прогнозу слід очікувати і тим менші втрати, які пов’язані з невизначеністю. ШНМ з РБФ мають ряд переваг у порівнянні з іншими математичними методами прогнозування.

ШНМ з РБФ мають більш гнучку структуру порівняно зі штучними нейронними мережами персептронного типу. Штучні нейромережі з радіально-базисними функціями є універсальними апроксиматорами, та в силу того, що в них присутній лише один нелінійний прихований шар, а налаштовуються параметри лінійного вихідного шару, для їх навчання можуть використовуватися стандартні процедури, яким притаманна висока швидкодія та фільтруючі засоби, що дуже важливо для задач опрацювання „зашумлених” даних [5]. У межах визначеної архітектури ШНМ з РБФ для зміни структури достатньо визначити кількість нейронів прихованого шару. Додаткову перевагу надає можливість зміни функції активації. Лише деякі незначні перетворення роблять можливим повністю змінити структуру ШНМ з РБФ, що дозволяє якнайкраще пристосувати обрану архітектуру, що розв’язується, та мінімізувати похибку навчання мережі (підвищити точність та якість прогнозу).

Ще одна не менш вагома перевага ШНМ з РБФ – експерт не залежить від вибору математичної моделі поведінки часового ряду. Побудова моделі ШНМ з РБФ відбувається адаптивно, під час навчання, без участі експерта. ШНМ з РБФ надаються конкретні приклади із бази даних, і вона сама налаштовується під ці дані.

Структура ШНМ з РБФ наведена на рисунку [7], з якого видно, що в мережі є вхідний шар, лише один прихований (радіально-базисний) шар і вихідний лінійний шар.



Узагальнена архітектура штучної нейронної мережі з радіально-базисними функціями [3, 4]

Навчання ШНМ з РБФ найчастіше проводиться за однокроковим або багатокроковим алгоритмом навчання. Для навчання мережі та формування її структури використано середовище MATLAB, оскільки воно дозволяє швидко опрацьовувати великі обсяги статистичних да них і забезпечено широким набором програм і функцій для проекту вання та дослідження штучних нейронних мереж даного типу.

Однокроковий алгоритм навчання створює ШНМ з РБФ похиб кою нульового значення на навчальній вибірці. Мережа вчиться за од ним кроком, структура мережі формується автоматично в процесі на вчання. Даний алгоритм формує штучну радіально-базисну нейроме режу з такими вагами і зміщеннями, що її виходи рівні бажаному ви ходу й структура даної штучної нейромережі формується таким чином, що кількість нейронів прихованого (радіально-базисного) рівня рівна числу елементів навчальної вибірки. Єдиною умовою, яку слід викона ти – це вибрати достатньо великим значення параметра впливу, але не настільки великим, щоб значення входів були рівнозначимими.

Навчання мережі за даним алгоритмом відбувається досить шви дко, проте структура моделі є загромадженою, не оптимізованою, оскільки

кількість нейронів прихованого шару рівна кількості елемен тів навчальної вибірки. Отже, за допомогою вищеописаної структури ШНМ з РБФ неможливо одержати якісний прогноз у випадку великих розмірів навчальної вибірки, що характерно для реальних задач.

При навчанні ШНМ з РБФ за багатокроковим алгоритмом на вчання структура штучної нейромережі радіального типу формується, використовуючи ітеративну процедуру, яка добавляє по одному ней рону на кожному кроці. Створюється дворівнева мережа. Перший рі вень складається з радіально-базисних нейронів і обчислює свої зва жені входи за допомогою функції евклідової відстані dist, а також свої питомі входи. Другий рівень складається з простих лінійних нейронів (y f x x = = ( )) і обчислює свій зважений вхід за допомогою відповід ної функції, а також свої питомі входи. Обидва рівні мають зміщення.
Функція dist виконує обчислення за формулою

$$d_i = \sqrt{\sum_j (x_j - w_{i,j})^2}\ ,\ i = \overline{1,S}\ ,$$

де wi, j , – елемент матриці ваг W; W – матриця ваг, що має розмір ність (S x R); xj – значення j-го входу, j R = 1, .

На початку роботи алгоритму радіально-базисний рівень не міс тить нейронів. Нейрони добавляються до прихованого шару до тих пір, поки сума квадратів середньоквадратичних похибок мережі не стане меншою за задане значення або не буде використано максимальної кількості нейронів. Обчислюється прогноз штучної нейронної мережі радіального типу. Знаходиться вхідний вектор з найбільшим значен ням середньоквадратичної похибки. Додається радіально-базисний нейрон з вагами, що дорівнюють цьому вектору. Ваги простого ліній ного рівня реорганізуються в такий спосіб, щоб мінімізувати серед ньоквадратичну похибку.

В процесі навчання ШНМ з РБФ за багатокроковим алгоритмом формується оптимальна структура нейромережі (з оптимальною кіль кістю нейронів прихованого рівня), яка даватиме точніший прогноз із мінімальною похибкою навіть тоді, коли об’єм навчальної вибірки значно великий. Структура моделі спрощується завдяки апроксимації вхідних даних із заданою точністю.

Отже, із проведеного аналізу випливає, що найбільш ефективним кількісним методом прогнозування є використання ШНМ з РБФ. Оскі льки для реальних задач сьогодення характерна велика кількість фак торів впливу, то в результаті дослідження встановлено, що саме алго ритм багатокрокового навчання доцільно застосовувати для навчання ШНМ з РБФ, який здатний працювати з великою вибіркою даних і в процесі навчання формувати структуру штучної нейронної мережі радіального типу, яка володіє прогностичними властивостями.

За час використання нейронних мереж у математичному моделюванні складних процесів і систем уже чітко виокремились ті задачі, для яких застосування саме цього інструментарію дає кращий результат порівняно з іншими типами математичних моделей. До таких задач відносяться задачі класифікації (категоризації), прогнозування, розпізнання образів тощо. Оцінка надійності позичальника банку відноситься якраз до задач класифікації. ерНайпоширенішим підходом до розв'язання даної задачі є використання логістичної регресії. Широке застосування цього типу моделі обумовлене простотою реалізації та зрозумілою інтерпретацією результатів. При побудові логістичної регресії для визначення скорингової оцінки кредитоспроможності позичальника-фізичної особи вхідними факторами слугують всі доступні для конкретної фінансової установи дані. До переліку таких даних включають як соціально-демографічні показники анкетних даних позичальників-фізичних осіб, так і статистичні дані, що пов'язані з кредитною історією та поточною платіжною дисципліною. Результатом застосування логістичної регресії є значення з інтервалу [0; 1], що інтерпретуються як імовірність невиконання позичальником кредитних зобов'язань. За умови забезпечення повноти та коректності вхідних статистичних чи анкетних даних можна очікувати достатньо точної оцінки кредитоспроможності позичальника в майбутньому.

Однак слід відмітити, що основним недоліком використання логістичної регресії є проблема виникнення мультиколінеарності (наявності тісного кореляційного або лінійного зв'язку між двома чи більше факторами, які включаються у модель). Проблема мультиколінеарності широко досліджена для випадку лінійної регресії, розроблено методи виявлення цього явища та способи зменшення його негативного впливу на модель. Проте для випадку логістичної регресії ця проблема досліджена недостатньо, хоча вона є такою ж актуальною, як і для лінійної регресії. За наявності мультиколінеарності дисперсія оцінок параметрів моделі регресії зростає пропорційно до їх значень, що призводить до не стійкості цих оцінок [1, с. 12]. Найгіршим проявом цієї проблеми є оцінки параметрів з протилежними знаками — тобто виникають випадки, коли знаки отриманих оцінок параметрів при певних факторах моделі протирічать фактичному напрямку впливу даного фактору на залежну змінну. Уникнути проблем, пов'язаних з мультиколінеарністю, можна за рахунок застосування для розв'язання задачі класифікації математичної моделі іншого типу, зокрема, нейронної мережі.

4.3 Мета і завдання дослідження

Метою роботи є побудова нейронних мереж різної архітекту ри для розв'язання задачі оцінки надійності позичальників фізичних осіб і проведення експериментального дослідження з вибору найадекватнішої з них.

Основним завданням роботи є вибір найкращої архітектури нейромережі та дослідження впливу зміни її конфігурації на точ ність класифікації позичальників за рівнем ризику дефолту за кредитними

зобов’язаннями. В роботі також досліджується пи тання використання комітету моделей, як альтернативи застосу вання лише одної нейронної мережі.

4.4 Виклад основного матеріалу

Штучна нейромережа виконує певну послідовність дій з перетворення вхідних даних у вихідні: кожен нейрон отримує сигнали через кілька вхідних з’єднань, що мають певну силу чи вагу, яка представлена відповідними коефіцієнтами; стан нейрону опису ється зваженою сумою його вхідних сигналів з доданим до неї пороговим значенням; цей розрахунок суматора нейрона перет ворюється за допомогою функції активації у вихідний сигнал. Найважливішою особливістю нейронної мережі є її здатність до навчання, що реалізується шляхом ітераційного налаштування зв’язків між її елементами, послідовно зменшуючи похибку мо делювання результативного показника.

Придатність нейронних мереж до розв’язання широкого спектру задач зумовила виникнення значного різноманіття видів штуч них нейромережевих структур. Вибір типу нейронної мережі залежить від поставленої задачі. У випадку, коли результуючий показник являє собою дискретну величину, що складається з двох чи більше груп даних, завдання нейромережі полягає у віднесенні вхідних даних до однієї з цих груп. В такій постановці маємо задачу класифікації чи категоризації даних. Процедура проведення класифікації в термінах теорії нейронних мереж описується таким чином: вихідний елемент нейромережі повинен видавати високий рівень активації в тому випадку, коли певне спостереження належить до відповідного класу, та низький рівень активації — у протилежному випадку. Наочне представлення результату моделювання нейронної мережі дає відповідна поверхня в багатовимірному просторі (де один вимір відводиться для відображення виходу мережі, а інші виміри вказують значення вхідних показників). Залежно від обраної архітектури нейромережі буде отримано поверхні різного виду та складності.

Для розв’язання задачі оцінки кредитоспроможності позичаль ників фізичних осіб у даній роботі застосовувались нейромережі трьох типів.

Найпростішою з них є лінійна мережа. Така мережа не містить проміжних шарів і для вихідного шару використовує лінійну фу нкцію активації (що є повним аналогом лінійної регресії). Для за дачі класифікації лінійна мережа моделює деяку гіперплощину таким чином, щоб вона розділяла між собою два класи.

Засоби пакету STATISTICA Neural Networks дозволяють зобра зити аналоги гіперплощин для тривимірного простору [9]. Такі поверхні мають назву поверхні відгуку. На рис. 1 схематично зображено поверхню відгуку для умовного прикладу розв’язання задачі класифікації.

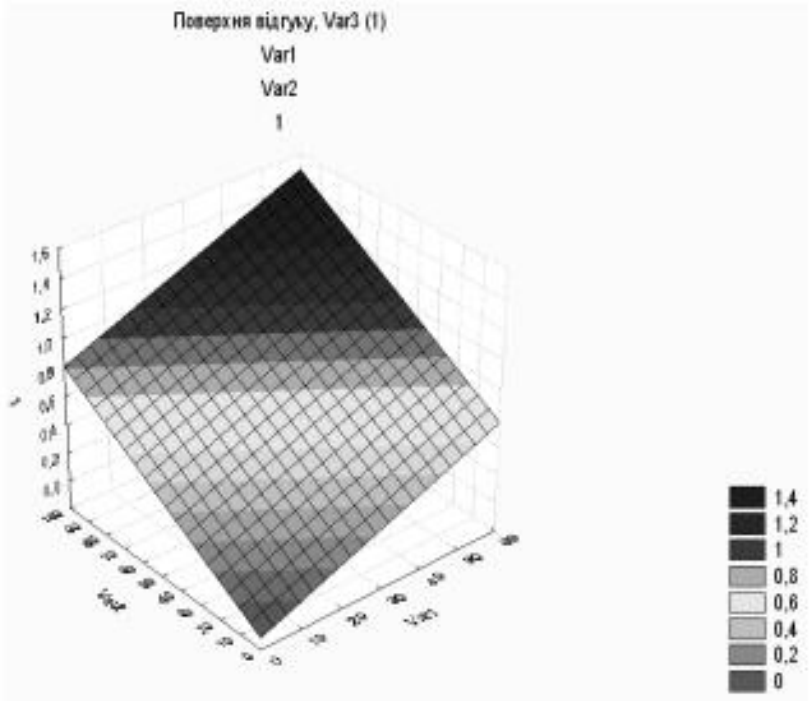


Рис. 1. Схематичне зображення поверхні відгуку для лінійної мережі

Вхідні показники відкладені в осях Var1 та Var2, вихідна змінна — Var3. Для наведеного на рис. 1 прикладу в задачі кла сифікації вихідна змінна приймає два значення: 0 та 1. Дана по верхня відображає, як зростає рівень активації вихідного нейрону лінійної нейромережі по відношенню лише до одного з класів, який позначено «1». Чим більшим є значення рівня активації ви хідного нейрону, тим більшою мірою відповідні вхідні дані ви значатимуть належність до класу «1». Очевидно, що такий прос тий спосіб проведення класифікації не завжди може дати прийнят ний результат.

При розв’язанні складних задач лінійну мережу застосовують для визначення найнижчого рівня якості моделі, що надалі дoз волить провести порівняння точності результатів моделювання та визначити доцільність використання складнішої архітектури нейромережі. Може виявитись, наприклад, що для задач, які не ма ють достатньої кількості даних, для навчання немає сенсу вико ристання складних моделей.

Однак у більшості випадків ефективним є застосування склад ніших нейромереж. Одним із найпоширеніших типів мереж є ба гатошаровий персептрон. В ньому нейрони вхідного шару слу гують для введення значень пояснюючих змінних. Кожен з ней ронів вхідного шару пов’язаний з усіма елементами першого прихованого шару. В свою чергу кожен нейрон із наступних при хованих шарів пов’язаний з усіма елементами попереднього ша ру, а шар вихідних нейронів також має зв’язок з усіма елемента ми останнього прихованого шару. За рахунок регулювання кіль кості шарів та елементів у кожному шарі нейромережа може мо делювати функцію довільного ступеня складності.

На рис. 2 наведено схематичне зображення багатошарового персептрону.

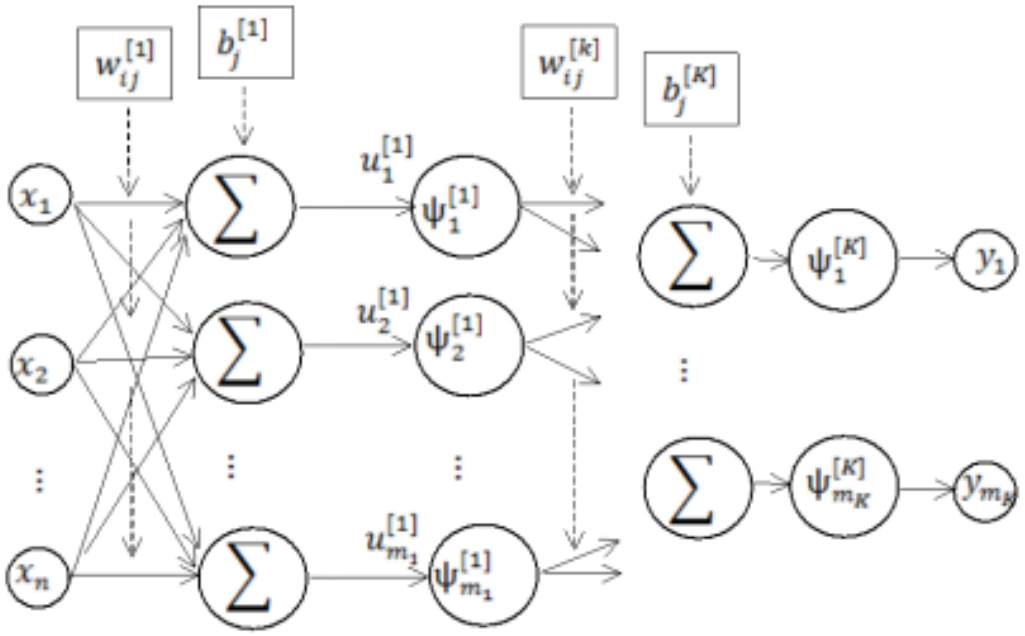


Рис. 2. Схематичне зображення багатошарового персептрону

Позначення на рис. 2: xi — значення i-го вхідного показника, [k] wij — значення вагового коефіцієнта, що поєднує i-ий нейрон (k – 1)-го шару (або i-ту вхідну змінну) з j-им нейроном k-го шару, k — номер шару, [k] bj — параметр зміщення суматора j-го нейро на k-го шару, [k] u j — розрахунок суматора j-го нейрона k-го шару (зважена сума вхідних сигналів), [k] Ψj — функція активації j-го нейрона k-го шару, yl — значення l-го вихідного показника, mk l =1, . Розрахунок суматора кожного нейрона здійснюється за формулою:

$$u_j^{[k]} = b_j^{[k]} + \sum_{i=1}^{n} w_{ij}^{[k]} x_i, \quad j = \overline{1, m_k}, \; k = \overline{1, K},$$

де mk — кількість нейронів у k-му шарі, K — кількість шарів пер септрону.

Значного рівня ефективності при розв’язанні задачі класифі кації багатошаровий персептрон досягає за рахунок здатності мо делювати більш складну поверхню відгуку порівняно з лінійною мережею. Чим складнішою є форма поверхні, тим точніше мо дель може здійснити класифікацію об’єктів. Хоча розрахунок су матора окремих нейронів у багатошаровому персептроні є ліній ною функцією вхідних значень, далі цей сигнал перетворюється за допомогою деякої нелінійної функції активації, наприк лад сигмоїдної, що приводить до утворення поверхонь з так зва ними «сигмоїдними нахилами». На рис. 3 наведено поверхню в тривимірному просторі, отриману за результатами розрахунків одного з нейронів першого прихованого шару багатошарового персептрону.

Поверхня на рис. 3 є функцією від двох вхідних змінних. Як що вхідних показників більша кількість, то буде отримано бага товимірний аналог такої поверхні. При зміні ваг міжнейронних зв'язків і параметрів зміщення суматора змінюється і поверхня відгуку. Збільшення значень ваг приведе до збільшення крутизни нахилу поверхні, а зміна параметрів зміщення суматора вплива тиме на орієнтацію поверхні в просторі.

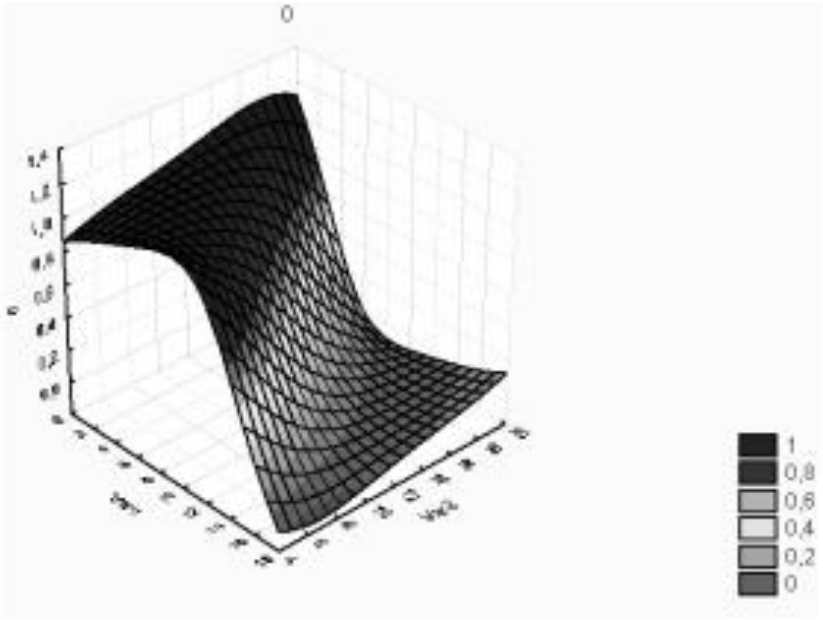


Рис. 3. Поверхня відгуку в тривимірному просторі за результатами моделювання одного нейрону

У багатошаровій мережі такі функції відгуку об'єднуються одна з одною шляхом послідовного отримання їх лінійних комбі націй і застосування деяких нелінійних функцій активації. На рис. 4 зображено поверхню відгуку, отриману за результатами розрахунків мережі з одним виходом та одним прихованим ша ром, який складається з двох нейронів. Кожен нейрон приховано го шару моделює свій сигмоїдний нахил, а поєднуючись разом у вихідному нейроні вони утворюють одну поверхню відгуку.

У процесі навчання багатошарового персептрону коригуються ваги міжнейронних зв'язків і параметри зміщення суматорів ней ронів. Відповідно, поверхні відгуку кожного з нейронів оберта ються в потрібному напрямку та змінюють крутизну нахилу, прилаштовуючись найкращим чином для проведення класифіка ції вхідних даних. Мережа, яка містить один прихований шар, будує кілька сигмоїдних нахилів. Їх кількість відповідає кількості нейронів у прихованому шарі, а вихідний нейрон комбінує з них спільну поверхню. На поверхні є вершини і западини, які відпо відають рівням належності вхідних даних (значення вхідних по казників відкладаються по відповідних осях) до певного класу. Якщо мережа містить два прихованих шари, то поверхня буде мати ще складнішу структуру і може містити більшу кількість вершин і западин. Зрозуміло, що використовуючи мережу такого типу

можна відтворити поверхню будь-якої складності, що дає можливість ефективно розв’язати більшість задач класифікації.

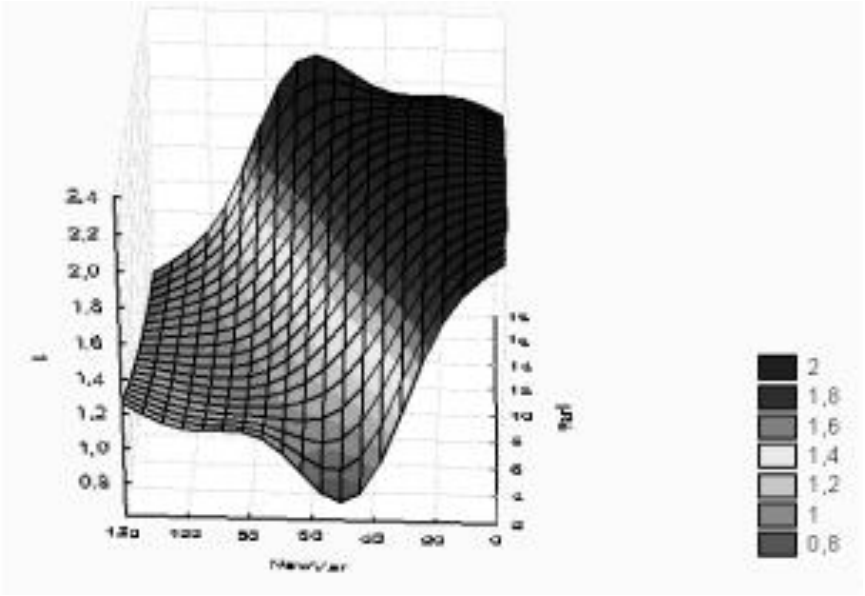


Рис. 4. Поверхня відгуку мережі з двома нейронами одного прихованого шару

Ще одним типом нейроархітектури є радіально-базисна ней ронна мережа. Функціонування такої мережі базується на іншому принципі формування вихідного сигналу. Цей принцип діє за аналогією справжніх нейронів специфічного типу, в яких вихід ний сигнал розповсюджується лише на певну обмежену область простору. Мережа, яка побудована на штучних нейронах з чітко вираженими локальними характеристиками, є альтернативою ба гатошаровим персептронам і має назву радіально-базисної ней ронної мережі. В такій мережі штучні нейрони характеризуються функціями активації, що радіально змінюються навколо обраного центру і приймають ненульові значення лише в певному околі цього центру. На рис. 5 наведено схематичне зображення струк тури радіально-базисної мережі.

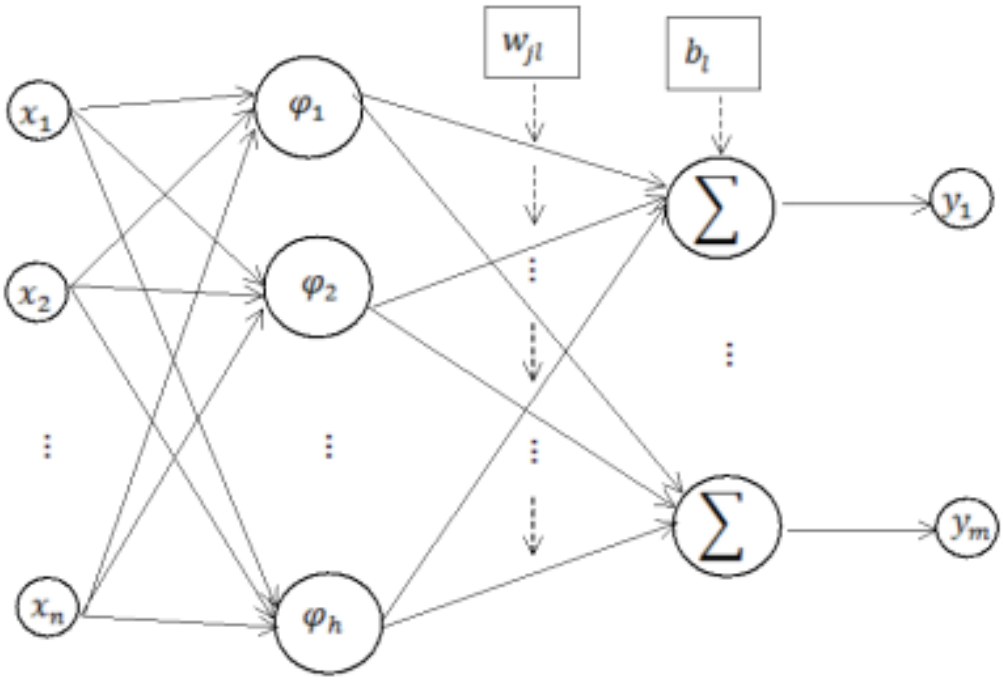


Рис. 5. Схематичне зображення структури радіально-базисної мережі

Позначення на рис. 5: xi — значення i-го вхідного показника, l =1,n , φ j — радіально-базисна функція, j =1,h , h — кількість нейронів прихованого шару, wjl — значення вагового коефіцієн та, що поєднує j-ий нейрон шару радіально-базисних функцій та l-ий нейрон вихідного шару, bl — параметр зміщення суматора l-го нейрона вихідного шару, yl — l-ий вихідний показник, l =1,m . Розрахунок виходу мережі здійснюється за формулою:

$$y_l = b_l + \sum_{j=1}^{h} w_{jl}\,\varphi_j, \quad l = \overline{1,m}.$$

Властивості такої мережі повністю визначаються радіально базисними функціями, які є основними компонентами у нейронах прихованого шару:

$$\varphi_j = \phi(\|x - c_j\|, \sigma_j).$$

Радіально-базисна функція є багатовимірною функцією, яка залежить від відстані j x – c між вхідним вектором x та власним Вибір архітектури нейромережі для розв’язання задачі… С. С. Савіна, В. П. Бень 135 центром cj , а також параметром σ j , який можна інтерпретувати як радіус [10]. Найпоширенішими радіально-базисними функція ми є гаусові функції, які приймають максимальне значення в центрі та монотонно спадають при віддаленні від нього.
За аналогією до поверхні відгуку в багатошаровому персепт роні, радіально-базисна мережа здійснює моделювання із засто суванням гіперсфер (рис. 6).

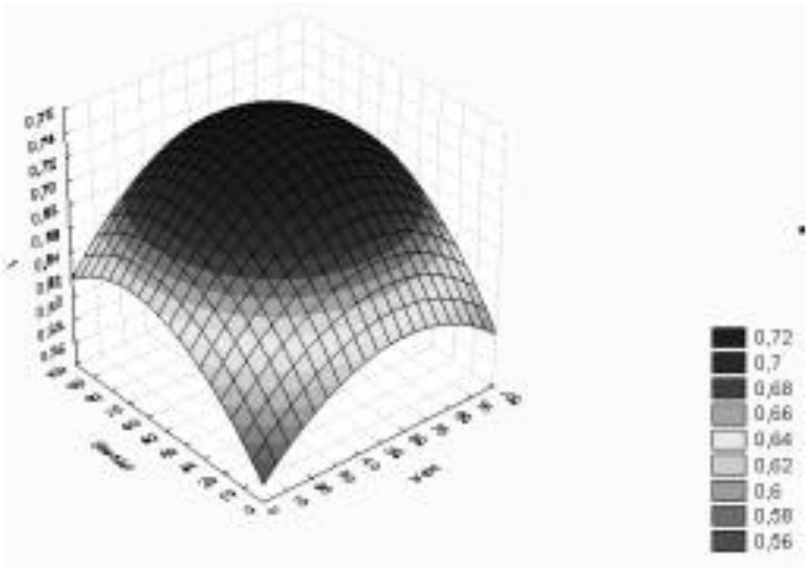


Рис. 6. Поверхня відгуку при використанні радіально-базисної нейромережі з одним нейроном

Поверхня відгуку радіальних елементів є дзвіноподібною функ цією з вершиною в центрі та зниженням до країв. Параметрами такої сфери є координати її центру в багатовимірному просторі cj та радіус σ j . Положення будь-якої точки у n-вимірному просторі визначається n числовими параметрами (тобто їх стільки, скільки ваг у звичайних нейронів у багатошаровому персептроні), тому координати центру радіальних нейронів є аналогом ваг міжней ронних звязків у персептроні. Відхилення від центру (радіус) ви значає ступінь кривизни гіперсфери. Радіально-базисна мережа містить один прихований шар з радіальних нейронів, кожен з яких моделює гаусову поверхню відгуку. Оскільки такі функції нелінійні, то для моделювання довільної функції немає потреби використовувати більше одного прихованого шару — потрібно лише взяти достатню кількість радіальних елементів. Для отримання розрахунку вихідної змінної в такій мережі визначають зважену суму гаусових функцій нейронів прихованого шару.

Радіально-базисні мережі мають ряд переваг порівняно з бага тошаровим персептроном: відсутність необхідності дослідження оптимальної кількості прихованих шарів (оскільки завжди вико ристовується лише один) і лінійна комбінація виходів нейронів прихованого шару у вихідному шарі. За рахунок таких переваг радіально-базисна мережа швидко навчається. З іншого боку, для моделювання складних вихідних функціональних залежностей радіально-базисна мережа вимагає більшої кількості елементів, ніж персептрон. Тому вибір типу мережі при розв’язуванні задачі класифікації залишається проблемою, вирішення якої вимагає проведення експериментальних розрахунків.

Крім вибору типу мережі постає завдання визначення опти мальної конфігурації мережі, що також вимагає проведення ряду експериментів. Спочатку обирається деяка початкова конфігура ція. Наприклад, багатошаровий персептрон з одним прихованим шаром і кількістю нейронів, за якої кількість параметрів мережі (ваг міжнейронних зв’язків між усіма нейронами кожної пари

су сідніх шарів і параметрів зміщення суматорів всіх нейронів) не перевищуватиме третину від кількості вхідних векторів. Далі проводиться ряд експериментів з різними конфігураціями. Крім того, для кожної окремої конфігурації також слід провести кілька експериментів, щоб переконатись, що процес навчання не привів до локального мінімуму похибки моделювання. Критерієм адек ватності нейромережі кожної нової конфігурації може слугувати відсоток правильно класифікованих прикладів з тестової множи ни. Якщо в результаті експерименту персептрон не демонструє задовільної точності класифікації, потрібно змінювати конфігу рацію, збільшуючи кількість нейронів або кількість прихованих шарів. Якщо має місце перенавчання мережі, слід зменшувати кількість нейронів у прихованих шарах чи кількість самих при хованих шарів. Аналогічні експерименти проводяться і з пошуку оптимальної конфігурації радіально-базисної мережі.

У даному дослідженні експерименти проводились в середо вищі пакету STATISTICA Neural Networks. В налаштуваннях паке ту, якщо не вказано конкретних значень, за якими здійснюється поділ повного масиву спостережень на навчальну та тестову ви бірки, для будь-якої архітектури нейромереж початковий масив даних ділиться на три частини у співвідношенні 2:1:1. Перша час- Вибір архітектури нейромережі для розв'язання задачі… С. С. Савіна, В. П. Бень 137 тина спостережень використовується як навчальна вибірка, на якій проводиться оптимізація параметрів мережі. Друга частина має назву контрольної вибірки, результати розрахунків за якою слугують для перевірки якості отриманої моделі. За результатами моделювання на контрольній вибірці приймаються висновки, чи прийнятною є якість мережі в поточній конфігурації, чи все-таки необхідно продовжити експерименти з корегування конфігурації для підвищення якості моделі. Зазначимо, що пакет STATISTICA Neural Networks передбачає можливість автоматичного налашту вання значної кількості мереж з різною конфігурацію та вибору найкращої з них. У такому випадку може виникати наступна проб лема. Оскільки розрахунки здійснюються автоматично, то в ре зультаті може бути обрано мережу з найвищим значенням ефек тивності на контрольній вибірці, що насправді не відповідатиме властивостям цієї мережі в загальнішому випадку. Тому виділя ється ще одна вибірка — тестова, яка використовується на за ключній стадії процесу аналізу адекватності моделі. Якщо ре зультати ефективності роботи мережі не мають значної різниці на контрольній і тестовій вибірках, то вважається, що нейромережа має потрібну якість навчання та демонструватиме достатньо стійкий результат при подальшому використанні.

Статистичним джерелом проведеного дослідження є дані з кре дитних заявок позичальників-фізичних осіб комерційного банку та відомостей щодо виконання ними зобов'язань за отриманими кредитами. Представлена інформація містить дані за 11 чинника ми та нараховує 2175 спостережень.

З 11 чинників три є якісними (рівень освіти, статус працюю чого, сфера діяльності), всі інші — кількісні показники (вік, стаж на останньому місці роботи, загальний стаж роботи, дохід, наяв ність депозитів, наявність виплачених у минулому кредитів, роз мір сім'ї, кількість дітей).

Розрахунки проведено на основі трьох типів архітектури ней ромереж: лінійна, багатошаровий персептрон і радіально-базис на мережа.
Як зазначалось вище, при використанні лінійної архітектури нейромережі можна отримати лише оцінку найнижчого рівня якості моделі. Конфігурація лінійної мережі не передбачає мож ливості вибору параметрів, крім кількості елементів у вхідних векторах. У роботі [11] обґрунтовано використання шести з оди надцяти факторів для проведення оцінювання кредитоспроможності позичальників-фізичних осіб, а саме: вік, стаж на останньо му місці роботи, загальний стаж, наявність депозитів, наявність виплачених у минулому кредитів і кількість дітей у сім'ї. У цьо му дослідженні було проведено розрахунки параметрів і показ ників ефективності лінійної мережі, для якої вхідними змінними спочатку обрано шість перелічених факторів, а потім поступово вводились всі наступні фактори згідно порядку зростання їх зна чущості. В табл. 1 наведено розрахунки точності класифікації по зичальників з навчальної, контрольної та тестової вибірок, отри мані на основі лінійної мережі за різної кількості вхідних факторів.

Таблиця 1

РОЗРАХУНКИ ЕФЕКТИВНОСТІ ЛІНІЙНОЇ МЕРЕЖІ ЗА РІЗНОЇ КІЛЬКОСТІ ВХІДНИХ ФАКТОРІВ

Архітектура мережі, кількість входів	Відсоток правильно класифікованих значень у навчальній вибірці, %	Відсоток правильно класифікованих значень у контроль-ній вибірці, %	Відсоток правильно класифікованих значень у тестовій вибірці, %
Лінійна, 6	60,5	40,0	46,0
Лінійна, 7	61,6	41,0	45,0
Лінійна, 8	61,7	42,0	44,0

З табл. 1 видно, що на навчальній вибірці коректно класифіку ється близько 60 % спостережень, на контрольній вибірці — 40 %, що є незадовільним рівнем точності класифікації. Підтвер дженням низького рівня якості моделі можуть слугувати майже аналогічні дані тестової вибірки. За даними табл. 1 маємо, що при введені додаткових факторів ефективність лінійної моделі прак тично не змінюється.

Як зазначалось вище, засоби пакету STATISTICA Neural Networks дозволяють автоматично здійснювати перебір значної кількості конфігурацій нейронних мереж. У даній роботі з метою пошуку найефективнішої мережі здійснювався перебір конфігу рацій, які включали від одного до восьми факторів. Для кожної нейромережі автоматично встановлювалась кількість нейронів проміжного шару, щоб загальна кількість параметрів моделі не перевищувала третину навчальної вибірки, забезпечуючи при цьому високу точність класифікації позичальників. У табл. 2 наведено результати розрахунків за мережами, які показали найви щу ефективність серед усіх проаналізованих конфігурацій.

Таблиця 2

РОЗРАХУНКИ ЕФЕКТИВНОСТІ ТРИШАРОВИХ ПЕРСЕПТРОНІВ РІЗНИХ КОНФІГУРАЦІЙ

Архітектура мережі, кількість входів, кількість нейронів прихованого шару	Відсоток правильно класифікованих значень у навчальній вибірці, %	Відсоток правильно класифікованих значень у контрольній вибірці, %	Відсоток правильно класифікованих значень у тестовій вибірці, %
Тришаровий персептрон, 8, 6	62,5	53,0	58,0
Тришаровий персептрон, 6, 4	60,7	54,0	59,0
Тришаровий персептрон, 5, 6	61,2	50,0	57,0
Тришаровий персептрон, 5, 7	61,6	53,0	58,0
Тришаровий персептрон, 5, 5	60,8	52,0	58,0
Тришаровий персептрон, 2, 6	52,3	45,0	51,0

У табл. 3 узагальнені дані проведеної класифікації із застосуванням найефективніших тришарових персептронів різної конфігурації, без поділу на підвибірки (одночасно по всьому масиву даних).

Аналіз отриманих у табл. 2 і 3 результатів дозволяє зробити висновок, що неможливо однозначно встановити зв'язок між кількістю вхідних факторів, кількістю нейронів проміжного шару та ефективністю моделі. Наприклад, коли мережа використовує п'ять вхідних чинників, а кількість нейронів проміжного шару змінюється від п'яти до шести, то на навчальній вибірці якість моделі зростає, тоді як на контрольній — зменшується. Аналогічний випадок спостерігається при збільшенні кількості вхідних факторів від шести до восьми: точність класифікації на навчальній вибірці зростає, а на контрольній зменшується.

Таблиця 3

УЗАГАЛЬНЕНІ РОЗРАХУНКИ ЕФЕКТИВНОСТІ ПЕРСЕПТРОНІВ ПО ВСЬОМУ МАСИВУ ДАНИХ

Архітектура мережі, кількість входів, кількість нейронів проміжного шару	Вихідна змінна		Правильно класифікованих, %
	Клас	Всього	
Тришаровий персептрон, 8, 6	0	1131	62,1
	1	1044	61,6
Тришаровий персептрон, 6, 4	0	1131	60,5
	1	1044	60,1
Тришаровий персептрон, 5, 6	0	1131	60,7
	1	1044	60,2
Тришаровий персептрон, 5, 7	0	1131	61,2
	1	1044	60,8
Тришаровий персептрон, 5, 5	0	1131	60,5
	1	1044	60,1
Тришаровий персептрон, 2, 6	0	1131	52,1
	1	1044	51,7

З метою ґрунтовнішого дослідження зв’язку конфігурації три шарового персептрону та якістю класифікації проведено експе риментальні розрахунки з фіксованою кількістю вхідних факто рів (використовувались дані за шістьма показниками) при зміні кількості нейронів прихованого шару від 1 до 12. Було проведено три серії експериментів та обчислено середні значення точності. Результати точності моделювання на навчальній та тестовій ви бірках представлені на рис. 7.

На рис. 7 ряд 1 описує точність класифікації на навчальній ви бірці, ряд 2 — на тестовій. Отримані результати свідчать, що точ ність класифікації на навчальній вибірці зростає зі збільшенням кількості нейронів прихованого шару, однак разом з тим зменшу ється точність класифікації на тестовій вибірці (за рахунок суттє вого зростання кількості параметрів моделі відбувається підлаш тування мережі під навчальну вибірку, що вказує на елементи прояву ефекту перенавчання). Найадекватнішими аналізованим вибіркам виявились структури персептронів, що містили 6 або 8 нейронів на прихованому шарі.

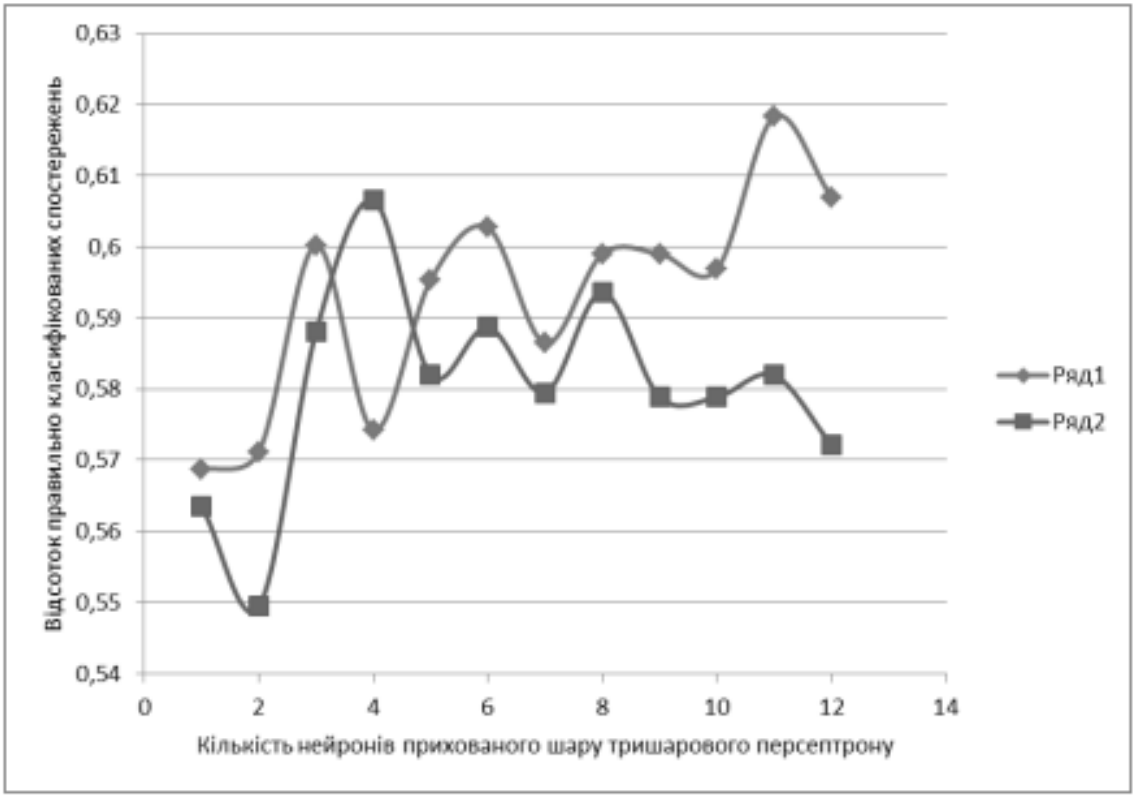


Рис. 7. Взаємозв’язок конфігурації тришарового персептрону та якості моделі

Результати дослідження з визначення оптимальної кількості вхідних факторів і нейронів проміжного шару при використанні мережі радіально-базисної архітектури наведені в табл. 4.

П’ять перших конфігурацій моделей демонструють результати найкращих за ефективністю радіально-базисних мереж на основі восьми факторів, наступні п’ять — на основі перелічених вище шести. Аналіз результатів моделювання підтверджує попередні висновки, отримані для лінійної мережі та персептрону: найвищу ефективність демонструють мережі, які містять шість вхідних факторів. Отже, при виборі конфігурації радіально-базисної ме режі варто обмежитись структурою з шістьма нейронами на вхід ному шарі.

Таблиця 4

РОЗРАХУНКИ ЕФЕКТИВНОСТІ РАДІАЛЬНО-БАЗИСНИХ МЕРЕЖ РІЗНИХ КОНФІГУРАЦІЙ

Архітектура мережі, кількість входів, кількість нейронів проміжного шару	Відсоток правильно класифікованих значень у навчальній вибірці, %	Відсоток правильно класифікованих значень у контрольній вибірці, %	Відсоток правильно класифікованих значень у тестовій вибірці, %
Радіально-базисна, 8, 67	64,7	45,0	45,0
Радіально-базисна, 8, 65	64,9	45,0	48,0
Радіально-базисна, 8, 80	66,2	45,0	47,0
Радіально-базисна, 8, 63	64,1	43,0	52,0
Радіально-базисна, 8, 67	64,7	45,0	45,0
Радіально-базисна, 6, 20	62,8	57,8	58,2
Радіально-базисна, 6, 24	62,1	59,1	59,1
Радіально-базисна, 6, 22	63,3	59,5	59,1
Радіально-базисна, 6, 41	62,9	59,9	59,9
Радіально-базисна, 6, 27	63,9	59,1	58,6

Окрім вибору архітектури та конфігурації нейромережі також важливим питанням, що впливає на якість моделювання, є спосіб формування навчальної вибірки. У описаному вище дослідженні ефективності моделей при автоматичному проведенні значної кіль кості експериментальних розрахунків навчальна, контрольна та тестова вибірки формуються в довільному порядку. Однак, оскіль ки результати розрахунку моделі можуть суттєво відрізнятись, залежно від обсягу та структури навчальної вибірки, були прове дені також дослідження впливу способу формування навчальної вибірки на точність моделювання кредитних ризиків.

Для задачі оцінки кредитоспроможності позичальників-фізич них осіб найвища точність моделі при перевірці на тестовій і ко нтрольній вибірках досягається при формуванні навчальної вибі рки, яка складається з даних, що відповідають надійним і нена дійним позичальникам у співвідношенні 1:1. Якщо загальний обсяг вибірки незначний, то необхідно найкращим чином здійс нити розподіл даних між навчальною та тестовою вибірками. Загальновживаним є розподіл, коли 70—80 % від загального маси ву даних складає навчальна вибірка, а 30—20 % використову ється для тестування моделі.

На основі вибірки, за якою проведено експериментальні роз рахунки в даній роботі, досліджувався взаємозв'язок між обсягом навчальної та тестової вибірок і ефективністю нейромереж різної архітектури. Враховуючи описані вище результати моделювання, для цього дослідження було обрано тришаровий персептрон з шістьма входами та кількістю нейронів проміжного шару 6, 8 та 10, а також радіально-базисна мережа на основі шістьох вхідних факторів і кількістю нейронів у прихованому шарі 82, 124, 200 та 335.

Розглянуто три варіанти розподілу повної сукупності даних на навчальну та тестову вибірки, а саме: навчальна вибірка склада лась з 1200, 1400 та 1600 спостережень — інші дані використову вались для тестування моделі. При цьому кожна начальна вибірка містила дані надійних і ненадійних позичальників у співвідно шенні 1:1. У табл. 5 представлено результати експериментів для нейромереж різної архітектури, в розрізі конфігурацій.

Таблиця 5

РОЗРАХУНКИ ЕФЕКТИВНОСТІ НЕЙРОМЕРЕЖ РІЗНОЇ АРХІТЕКТУРИ ТА КОНФІГУРАЦІЙ ЗАЛЕЖНО ВІД ОБСЯГУ НАВЧАЛЬНОЇ ВИБІРКИ

навчальна вибірка 1200 спостережень		
кількість нейронів проміжного шару	відсоток правильно класифікованих спостережень у навчальній вибірці, %	відсоток правильно класифікованих спостережень у тестовій вибірці, %
тришаровий персептрон		
6 8 10	65,0 65,1 65,8	52,1 47,0 50,0
радіально-базисна мережа		
82 124 200 335	69,3 71,1 59,3 62,9	53,5 51,2 52,1 49,9

навчальна вибірка 1400 спостережень		
кількість нейронів проміжного шару	відсоток правильно класифікованих спостережень у навчальній вибірці, %	відсоток правильно класифікованих спостережень у тестовій вибірці, %
тришаровий персептрон		
6 8 10	60,2 64,7 66,9	55,2 48,4 50,1
радіально-базисна мережа		
82 124 200 335	71,9 59,9 63,4 62,9	48,9 54,1 45,9 49,0
навчальна вибірка 1600 спостережень		
кількість нейронів проміжного шару	відсоток правильно класифікованих спостережень у навчальній вибірці, %	відсоток правильно класифікованих спостережень у тестовій вибірці, %
тришаровий персептрон		
6 8 10	62,9 64,3 65,4	49,4 43,7 45,0
радіально-базисна мережа		
82 124 200 335	59,1 60,6 55,4 56,2	48,3 44,0 46,1 46,4

Аналіз даних табл. 5 вказує, що найкращі результати на на вчальній і тестовій вибірках дає перший варіант розбивки (з найменшим обсягом навчальної вибірки), найгірші — третій ва ріант (з найбільшою кількістю спостережень у навчальній ви бірці). Другий варіант незначно поступається першому. Тобто, навчальна вибірка для побудови моделі дає найкращий результат, коли її обсяг складає 55—65 % від загального масиву спостережень.

Проведення великої кількості експериментів для виявлення оптимального розподілу спостережень на навчальну та тестову вибірку вимагає значних витрат часу і не завжди буде виправ дано. Як видно з табл. 5, перший і другий варіант формування навчальної вибірки дає схожі результати. Тому доцільно було б обрати таку архітектуру та конфігурацію мережі, яка б давала прийнятну якість моделі для більшої кількості можливих варіан тів змін умов її навчання. Однак, аналіз даних табл. 5 дає підста ви для висновку, що обрати єдину найефективнішу модель як за архітектурою, так і за конкретною конфігурацією неможливо. Так, наприклад, найвищу точність на навчальній вибірці для

ва ріанту 1200 спостережень демонструє радіально-базисна мережа з проміжним шаром із 124 нейронів, хоча на тестовій вибірці во на показує середню ефективність. Найкращий результат на тес товій вибірці для варіанту 1200 спостережень демонструє радіа льно-базисна мережа з проміжним шаром у 82 нейрони, хоча і дещо гірший на навчальній, ніж попередня мережа.
Якщо зупинитись на використанні лише однієї радіально-ба зисної мережі, залишається відкритим питання оптимізації кіль кості нейронів проміжного шару. Крім того, за даними табл. 5 маємо, що для другого та третього варіантів обсягу навчальної вибірки обрані дві моделі вже не є найкращими. Наприклад, для останнього варіанту розбиття масиву даних (1600 спостережень для навчальної вибірки) найточніший результат дає тришаровий персептрон з 6 або 8 нейронами на проміжному шарі. В цілому, з аналізу табл. 5 можна зробити висновок, що дещо кращі резуль тати моделювання кредитного ризику демонструють нейромере жі радіально-базисної архітектури. Проте, за неможливості здійс нювати постійні експериментальні розрахунки для визначення як оптимальних обсягів навчальної та тестової вибірок, так і вибору оптимальної конфігурації мережі та її налаштування, доцільно звернутись до ідеї застосування з метою вирішення задачі класи фікації одночасно кількох моделей нейромереж.
В поставленій у дослідженні задачі класифікації позичальни ків за рівнем ризику дефолту за кредитними зобов'язаннями до цільно використовувати композицію кількох нейромереж, що дасть змогу компенсувати помилки окремих моделей. Початкова задача вирішується кількома класифікаційними моделями, які розбивають вхідний простір на кілька підпросторів. Загальне рі шення є поєднанням рішень окремих моделей. Комбінацію таких моделей називають комітетом експертів (асоціативною маши ною, ансамблем) [12]. Такий підхід дозволяє поєднати «знання» кожного окремого «експерта» в загальне рішення, яке має пріоритет над рішенням окремого «експерта». Вважається, що використання комітету мо делей дозволить підвищити загальну точність моделювання.
Процедура формування рішення при використанні кількох ек спертів-моделей залежить від того, яким чином комбінуються кі лька експертних оцінок для отримання рішення комітету. Роз різняють дві категорії комітетів експертів [10, с. 458—459]:
1. Статичні структури. В такій категорії результати рішень окремих експертів об'єднуються за допомогою деякого механіз му, що не враховує вхідний сигнал (усереднення за ансамблем, підсилення).
2. Динамічні структури. У цій категорії вхідний сигнал безпо середньо враховується в механізмі об'єднання висновків експер тів (змішання рішень експертів, ієрархічне об'єднання висновків експертів).
Кожна із структур у свою чергу налічує кілька методів отри мання загального рішення. Найпростішим методом отримання рішення комітету, як статичної структури, є різні способи усере днення по ансамблю. Наприклад, такий:

$$Y(x) = \frac{1}{N} \sum_{i=1}^{N} y_i(x),$$

де x — вектор вхідних даних,yi — результат розрахунку i-ї ней ромережі (оцінка i-го експерта), N — кількість експертів у комі теті, Y — загальний результат роботи комітету моделей.

За наведеної процедури об'єднання результатів розрахунків кожної нейромережі маємо аналог простого голосування експер тів. У даній роботі класифікація позичальників-фізичних осіб на основі вказаного підходу відбувається таким чином. Кожна мо дель (експерт) проводить класифікацію позичальника, присвою ючи йому значення 0 чи 1. За всіма моделями знаходиться серед нє значення для кожного позичальника, і якщо значення 2 ( ) N Y x ≥ (тобто більшістю експертів для нього встановлено значення 1), то позичальник класифікується як ненадійний. У протилежному ви падку він вважається надійним. За аналізом показників ефективності моделей у табл. 5 для утворення комітету було обрано три нейромережі — дві з радіа льно-базисною архітектурою (82 та 124 нейрони на прихованому шарі) та тришаровий персептрон з 6 нейронами проміжного ша ру. Радіально-базисні нейромережі демонструють кращі резуль тати для меншого обсягу навчальної вибірки; персептрон більш ефективний, коли навчальна вибірка збільшується.

У табл. 6 подано узагальнені результати класифікації за обра ними для комітету моделями для навчальної вибірки з 1200 спо стережень.

Таблиця 6

РОЗРАХУНКИ ТОЧНОСТІ КЛАСИФІКАЦІЇ ПОЗИЧАЛЬНИКІВ ОКРЕМИМИ НЕЙРОМЕРЕЖАМИ З КОМІТЕТУ МОДЕЛЕЙ НА НАВЧАЛЬНІЙ ВИБІРЦІ З 1200 СПОСТЕРЕЖЕНЬ

Архітектура мережі, кількість входів, кількість нейронів проміжного шару	Приклади з загального масиву даних		Правильно класифікованих, %
	Клас	Всього	
Тришаровий персептрон, 6, 6	0	1131	56,9
	1	1044	61,7
Радіально-базисна, 6, 82	0	1131	59,9
	1	1044	64,8
Радіально-базисна, 6, 124	0	1131	59,8
	1	1044	64,8

Табл. 6 підтверджує отриманий вище результат, що кращу якість при оптимізації на навчальній вибірці з 1200 спостережень дають мережі радіально-базисної архітектури. Вони продемонст рували точність класифікації ненадійних

позичальників на зага льному масиві спостережень близько 65 %, тоді як тришаровий персептрон забезпечив 62 %.
У табл. 7 наведено результати розрахунків для комітету мо делей у всіх трьох варіантах формування обсягів навчальної ви бірки.

Таблиця 7

РОЗРАХУНКИ ТОЧНОСТІ КЛАСИФІКАЦІЇ ПОЗИЧАЛЬНИКІВ КОМІТЕТАМИ МОДЕЛЕЙ

Обсяг навчальної вибірки	Приклади з загального масиву даних		Правильно класифікованих, %
	Клас	Всього	
1200 спостережень	0	1131	59,2
	1	1044	64,8
1400 спостережень	0	1131	57,0
	1	1044	63,4
1600 спостережень	0	1131	56,1
	1	1044	62,3

Дані табл. 7 ілюструють роботу комітету моделей при зміні обсягів навчальної та тестової вибірок. Аналіз вказує, що як для окремих нейромереж, так і для комітету спостерігається тенден ція зниження точності класифікації зі зростанням обсягу навчаль ної вибірки. Однак, за наведеними загальними даними ефектив ності комітету моделей видно, що результат роботи комітету є стійкішим. У порівнянні з даними табл. 6 можна бачити, що для окремих нейромереж точність класифікації ненадійних позичаль ників варіюється від 62 % до 65 %, тоді як для комітету, сфор мованому з цих мереж, відповідне значення не є середнім, а на ближається до верхньої межі оцінки кращою з нейромереж. Тоб то можна вважати, що завдяки поєднанню кількох мереж-експер тів зміни умов проведення розрахунків не матимуть різкого впливу на зміну точності класифікації, що підвищує робастність системи загалом.
Якщо порівняти дані табл. 7 з показниками табл. 5, то можна побачити, що ефективність окремих нейронних мереж виявля лась дещо вищою за комітет моделей. Разом з тим, результат роз рахунків комітету є стабільнішим — точність діагностування не надійних позичальників становила 64 %, 63 % і 62 %, відповід но до трьох варіантів обсягів навчальних вибірок. Зазначимо, що саме точність визначення ненадійних позичальників є найважливішою характеристикою моделі оцінки кредитоспроможності по зичальників-фізичних осіб.

Проведено дослідження з вибору найадекватнішої ар хітектури нейромережі для розв’язання задачі оцінки кредито спроможності позичальників-фізичних осіб. У процесі проведен ня модельних експериментів на наявних статистичних даних вищу ефективність продемонстрували радіально-базисні нейрон ні мережі, однак при зміні умов навчання моделі кращі результа ти моделювання може проявляти персептрон. У результаті отри мано висновок, що вибір архітектури мережі доцільно проводити в рамках кожної окремої задачі з урахуванням наявних статисти чних даних. Окремим етапом дослідження став пошук оптималь ної конфігурації для обраних архітектур нейромереж. Дослідже но також взаємозв’язок між структурою навчальної вибірки та ефективністю моделювання.

Узагальнюючи результати всіх етапів дослідження встановле но, що неможливо однозначно обрати нейромережу, яка буде найкращим чином здійснювати оцінку кредитоспроможності по зичальників. На основі аналізу адекватності моделей обрано три конфігурації нейромереж, які об’єднано у комітет моделей, що надало змогу компенсувати можливі помилки класифікації окре мими моделями та підвищити загальну точність оцінки ризику невиконання позичальником кредитних зобов’язань.

У результаті проведення експериментів одержано підтвер дження доцільності використання комітету моделей, оскільки це дає можливість отримати стійкіші та вищі результати моделю вання, не витрачаючи додаткових ресурсів на визначення ряду параметрів, які необхідно обов’язково встановлювати при вико ристанні однієї нейромережі: визначення архітектури мережі, яка найкращим чином моделює вихідні дані; конструювання конкрет ної конфігурації мережі; вибір оптимальних обсягів навчальної та тестової вибірок. Крім того, слід враховувати, що зі зміною одно го з вищенаведених параметрів при конструюванні окремої ней ронної мережі всі інші також будуть змінюватися. Тому для змен шення обсягу експериментальних досліджень достатньо сформу вати комітет нейромереж з різною архітектурою та конфігура цією на основі усереднених значень їх розрахунків. Результат роботи такого комітету дасть стійкий і прийнятний рівень точності класифікації позичальників за їх кредитним ризиком.

Результати проведеного дослідження можуть бути використа ні банківськими та іншими фінансовими установами, зацікавле ними в адекватній процедурі оцінки кредитоспроможності фізич них осіб.

ВИСНОВКИ

					КНУ.РБ.123.16.35.В		
Змн.	Арк.	№ документа	Підпис	Дата			
Розробив		Лебідь			ВИСНОВКИ	Літера	Аркуш	Аркушів
Перевірив		Купін						
						КІ-21М		
Н.контроль		Кузнєцов						
Затвердив		Купін						

Обчислювальна система на основі використання принципу роботи нейромереж - нейрокомп’ютер - може бути представлений не тільки у вигляді окремого «системного блоку», як звичайний ПК, а й у вигляді плати розширення – нейрообчислювачі та нейроприскорювачі, які під’єднуються до плати розширення звичайного ПК, наприклад до шин PCI, USB, ISA, LPT та ін.. Апаратна реалізація нейрокомп’ютера має свої недоліки, а саме дороговизна розробки, та неможливість реалізації окремих її елементів. Найбільш відомим нейрокомп’ютером є Synaps1.

Серед найбільш поширених нейроалгоритмів виділяють алгоритми навчання персептрону, статичного навчання, Уідроу-Хоффа, ВР. В основі в основі кожного з цих алгоритмів полягає сума добутків, а це означає що будь-який з вище зазначених алгоритмів можна, наприклад розпаралелити.

Серед найбільш потужних нейрочипів та найбільш раціональним з категорії ціна/якість є процесор NeuroMatrix NM6403;

Розробка обчислювальної системи (нейрокомп’ютера) на основі нейромережі РБФ відбувалась на базі модулю МС4.31, фірми «Модуль»;

У приведених схемах розроблюваного нейрокомп’ютера головною особливістю є те, що в основі кожної системи є кластер;

Найбільш продуктивнішим нейрокомп’ютером є схема представлена на рис.2.6, котра характеризується незалежністю виконання задач, збільшенням продуктивності обчислень, та можливістю мережевого доступу до «нейронного кластеру».

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

					КНУ.РБ. 123.16.35.СВД		
Змн.	Арк.	№ документа	Підпис	Дата			
Розробив		Лебідь			СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	Літера	Аркуш	Аркушів
Перевірив		Купін						
						КІ-21М		
Н.контроль		Кузнєцов						
Затвердив		Купін						

1. Проблемы построения и обучения нейронных сетей / под ред. А.И.Галушкина и В.А.Шахнова. - М. Изд-во Машиностроение. Библиотечка журнала Информационные технологии №1. 1999. 105 с.
2. А.И.Галушкин Некоторые исторические аспекты развития элементной базы вычислительных систем с массовым параллелизмом (80- и 90- годы) // Нейрокомпьютера, №1. 2000. - С.68-82
3. А.Н.Горбань, Д.А.Россиев Нейронные сети на персональном компьютере. - Новосибирск: Наука. Сибирская издательская фирма РАН, 1996. - 276 с.
4. А.И. Власов. Аппаратная реализация нейровычислительных управляющих систем //Приборы и системы управления - 1999, №2, С.61-65.
5. CitFORUM[Електронний ресурс]:Нейрокомп’ютери: Архітектура та реалізація.- електр. дані.- M:CitForum,2008.- режим доступу: http://citforum.univ.kiev.ua/hardware/neurocomp/index.shtml, вільний.
6. ChipInfo[Електронний ресурс]: Нейрокомп’ютери архітектура та реалізація. Апаратна реалізація нейрообчислювачів.- електор. дані. – M.:ChipInfo, 2008.- режим доступу: http://www.chipinfo.ru/literature/chipnews/200008/12.html, вільний.
7. Новости IT[Електронний ресурс]: Нейрокомп’ютери. Новини. -електр. дані.- M.: IU4, 2009, режим доступу: http://neurnews.iu4.bmstu.ru, вільний.
8. Модуль[Електронний ресурс]: Новини. -електр. дані.- M.:Модуль, 2009.- режим доступу: http://module.ru/news.html, вільний
9. РФЯЦ-ВНИИТФ Електронний ресурс]:Наука та технології. -електр. дані.- M.:VNITF, 2007.- режим доступу: http://www.vniitf.ru/, вільний.
10. Вычислительная физика, прикладные сетевые исследования и ChANT[Електронний ресурс]: А.Ю. Довженко, С.А. Крашаков Параллельная нейронная сеть с удаленным доступом на базе распределенного кластера
11. ЭВМ.- електр. дані.-M: Comphys, 2008.- режим доступу: http://www.comphys.ru/Articles/cacr2001.htm, вільний.

12. Нейрокомп’ютераы - архитектура и реализация[Електронний ресурс]: Элементная база нейровычислителей.- електр. дані.-М: "В помощь Веб-Мастеру", 2009.- режим доступу: http://wm-help.net/books-online/book/25102/25102-0.html, вільний.
13. Модуль[Електронний ресурс]:NeuroMatrix NM 6403.- електр. дані.-М:Модуль, 2005.- режим доступу: http://www.module.ru/products/nm/nm6403.shtml , вільний.
14. Модуль[Електронний ресурс]: Модуль МС4.31.- електр. дані.-М:Модуль, 2005.- режим доступу: http://www.module.ru/ruproducts/dspmod/mc431.shtml, вільний.
15. Вікіпедія[Електронний ресурс]: Екологія. -електр. дані.-, - М.:Wikipedia, 2009. –режим доступу: http://www.bg.wikipedia.org/wiki/Екология , вільний.
16. Стиль[Електронний ресурс]:Екологія. -електр. дані.-, М.: ECO, 2009.- режим доступу: www.eco-style.com.ua, вільний.
17. Шпаковский Г.И. Организация параллельных ЭВМ и суперскалярных процессоров. Учебное пособие./ Г.И. Шпаковский. - Мн.: Белгосуниверситет, 1996.
18. Рихтер Д. Windows для профессионалов: Создание эффективных Win32-приложений с учетом специфики 64-разрядной версии Windows/ Д. Рихтер. – М.: Русская редакция, 2001. – 752с.
19. Харт. Дж.В. Системное программирование в среде Win32/ Дж. В. Харт. – М.: Вильямс, 2001. – 46 4 с.
20. Давидов Е.Г. „Исследование операций”.-М.: Вища школа, 1990.-378с.
21. Денисов А.А., Колесников Д.Н. Теория больших систем управления.- Л.: Энергоиздат.Ленингр. отд-ние, 1982.- 287с
22. Молчанов А.А. Моделирование и проектирование сложных систем.- К.: Выща школа, 1988.- 359с.

ДОДАТКИ

					КНУ.РБ.123.16.35.Д		
Змн.	Арк.	№ документа	Підпис	Дата			
Розробив		Лебідь			ДОДАТКИ	Літера	Аркуш	Аркушів
Перевірив		Купін						
						КІ-21М		
Н.контроль		Кузнєцов						
Затвердив		Купін						

Додаток А. Лістинг файлу моделювання роботи нейромережі

```
P = -1:.1:1;
T = [-.9602 -.8770 -.0729 .3771 .6405 .6600 .4609 .1336 …
      -.2013 -.4344 -.5000 -.3930 -.1647 .0988 .3072 .3960 …
     .3449 .1816 -.0312 -.2189 -.3201];
GOAL = 0.01;
SPREAD = 1;
net = newrb(P, T,GOAL, SPREAD); % Создание сети
net.layers{1}.size %Число нейронов в скрытом слое
NEWRB, neurons = 0, SSE = 3.69051

figure(1), clf,
plot(P,T,’sr’,’MarkerSize’,8,’MarkerFaceColor’,’y’)
hold on;
X = -1:.01:1;
Y = sim(net, X); %Моделирование сети

>> P = -1:.1:1;
T = [-.9602 -.5770 -.0729 .3771 .6405 .6600 .4609 .1336 -.2013 -.4344 -.5000 -
.3930 -.1647 .0988 .3072 .3960 .3449 .1816 -.0312 -.2189 -.3201];
GOAL = 0.01;
SPREAD = 0.01;
net = newrb(P,T,GOAL,SPREAD);
net.layers{1}.size % Число нейронов в скрытом слое
figure(1), clf,
plot(P,T,'sr','MarkerSize',8,'MarkerFaceColor','y')
hold on;
X = -1:.01:1;

Y = sim(net,X); % Моделирование сети
plot(X,Y,'LineWidth',2), grid on
```

Введение Для решения дифференциальных уравнений в частных производных (ДУЧП) эффективно применять нейронные сети. В настоящее время большой интерес вызывают

методы решения ДУЧП с применением радиально-базис ных функций (RBF) [1]. Эти методы

```
//=============================================================================
=
// Custom Solution Wizard - NSNetwork Class Implementation
//
//-----------------------------------------------------------------------------
-
// This file is provided to demonstrate how classes can be constructed to
// encapsulate the functions in a Custom Solution Wizard generated DLL.
// The classes in this file can be used without modication, but are not
// required for interaction with generated DLLs.
//=============================================================================
=

//=============================================================================
=
// Includes
//=============================================================================
=
#include "stdafx.h"
#include "NSNetwork.h"

//=============================================================================
=
//=============================================================================
=
// CLASS: NSNetwork
//=============================================================================
=
//=============================================================================
=

//-----------------------------------------------------------------------------
-
// CLASS: NSNetwork               FUNCTION: <constructor> (LPCSTR)
//-----------------------------------------------------------------------------
-
NSNetwork::NSNetwork(LPCSTR dllPathName)
{
      //-------------------------------------------------------------------
-------
      // Load the neural network DLL.
      //-------------------------------------------------------------------
-------
      m_hDLL = LoadLibrary(dllPathName);
      m_pNetworkInstance = 0;

      //-------------------------------------------------------------------
-------
      // If the DLL load was successful, get the addresses of the DLL
functions.
      //-------------------------------------------------------------------
-------
      if (m_hDLL)
      {
            //-------------------------------------------------------------
------------
            // Load functions common to both RECALL and LEARNING DLLs.
            //-------------------------------------------------------------
------------
            m_NSCreateNetwork = (NSCreateNetwork)GetProcAddress(m_hDLL,
"createNetwork");
```

```
            m_NSLoadWeights = (NSLoadWeights)GetProcAddress(m_hDLL,
"loadWeights");
            m_NSSaveWeights = (NSSaveWeights)GetProcAddress(m_hDLL,
"saveWeights");
            m_NSRandomizeWeights = (NSRandomizeWeights)GetProcAddress(m_hDLL,
"randomizeWeights");
            m_NSResetNetwork = (NSResetNetwork)GetProcAddress(m_hDLL,
"resetNetwork");

            //-----------------------------------------------------------------
------------
            // Verify functions common to both RECALL and LEARNING DLLs
loaded correctly.
            //-----------------------------------------------------------------
------------
            if (!m_NSCreateNetwork ||
                  !m_NSDestroyNetwork ||
                  !m_NSGetInputOutputInfo ||
                  !m_NSGetResponse ||
                  !m_NSGetSensitivity ||
                  !m_NSLoadWeights ||
                  !m_NSSaveWeights ||
                  !m_NSRandomizeWeights ||
                  !m_NSResetNetwork)
            {
                  FreeLibrary(m_hDLL);
                  m_hDLL = 0;
            }
      }
}

//-------------------------------------------------------------------------
-
// CLASS: NSNetwork               FUNCTION: <destructor>
//-------------------------------------------------------------------------
-
NSNetwork::~NSNetwork()
{
      //-----------------------------------------------------------------
-------
      // If the network instance is still set, release it.
      //-----------------------------------------------------------------
-------
      if (IsInitialized())
      {
            m_NSDestroyNetwork(m_pNetworkInstance);
            m_pNetworkInstance = 0;
      }

      //-----------------------------------------------------------------
-------
      // Release the neural network DLL if it is loaded.
      //-----------------------------------------------------------------
-------
      if (IsLoaded())
      {
            FreeLibrary(m_hDLL);
            m_hDLL = 0;
      }
}
                                                                        //
```

```
//-------------------------------------------------------------------------
//ÊËÀÑÑ: NSNetwork               ÔÓÍÊÖÈß: GetInputs()
//-------------------------------------------------------------------------
-
// Âåðíèòå ÷èñëî âõîäîâ äëÿ ãåíåðèðóåìîãî DLL èëè -1 íà îòêàçå.
//-------------------------------------------------------------------------
-
int NSNetwork::GetInputs()
{
      int numInputs;
      int numOutputs;
      if (!IsInitialized())
            return -1;
      if (m_NSGetInputOutputInfo(m_pNetworkInstance,numInputs,numOutputs))
            return -1;
      return numInputs;
}

//-------------------------------------------------------------------------
-
// CLASS: NSNetwork              FUNCTION: GetOutputs()
//-------------------------------------------------------------------------
-
// Return the number of outputs for the generated DLL or -1 on failure.
//-------------------------------------------------------------------------
-
int NSNetwork::GetOutputs()
{
      int numInputs;
      int numOutputs;
      if (!IsInitialized())
            return -1;
      if (m_NSGetInputOutputInfo(m_pNetworkInstance,numInputs,numOutputs))
            return -1;
      return numOutputs;
}

//-------------------------------------------------------------------------
-
// ÊËÀÑÑ: NSNetwork              ÔÓÍÊÖÈß: GetInputOutputInfo(int,int)
//-------------------------------------------------------------------------
-
// Ïîëó÷èòå ÷èñëî âõîäîâ è âûõîäîâ äëÿ ãåíåðèðóåìîãî DLL.
// Âîçâðàòû 0 íà óñïåõ, -1, åñëè íå èíèöèàëèçèðîâàíî, èëè êîä îøèáêè èç
DLL.
//-------------------------------------------------------------------------
-
//-------------------------------------------------------------------------
-
int NSNetwork::GetInputOutputInfo(int& numInputs, int& numOutputs)
{
      if (!IsInitialized())
            return -1;
      return
m_NSGetInputOutputInfo(m_pNetworkInstance,numInputs,numOutputs);
}

//-------------------------------------------------------------------------
-
// ÊËÀÑÑ: NSNetwork              ÔÓÍÊÖÈß: GetResponse(int,double*,double*)
//-------------------------------------------------------------------------
-
// Ïîëó÷èòå íåðâíûé îòâåò ñåòè ê íàâîðó íà÷àëüíûõ äàííûõ.
```

```
}

//---------------------------------------------------------------------------
-
// ÊËÀÑÑ: NSNetwork               ÔÓÍÊÖÈß:
GetSensitivity(int,double*,double*,double)
//---------------------------------------------------------------------------
-
// Ïîëó÷èòå íåðâíóþ ÷óâñòâèòåëüíîñòü ñåòè ê íàáîðó íà÷àëüíûõ äàííûõ.
// Âîçâðàòû 0 íà óñïåõå, -1, åñëè íå èíèöèàëèçèðîâàíî, èëè êîä îøèáêè èç
DLL.
//---------------------------------------------------------------------------
-
int NSNetwork::GetSensitivity(int exemplars, double* inputData, double*
sensitivityData, double dither)
{
      if (!IsInitialized())
            return -1;
      return
m_NSGetSensitivity(m_pNetworkInstance,exemplars,inputData,sensitivityData,di
ther);
}

//---------------------------------------------------------------------------
-
// ÊËÀÑÑ: NSNetwork               ÔÓÍÊÖÈß: LoadWeights(LPCSTR)
//---------------------------------------------------------------------------
-
// Çàãðóçèòå âåñîâ.
// Âîçâðàòû 0 íà óñïåõå, -1, åñëè íå èíèöèàëèçèðîâàíî, èëè êîä îøèáêè èç
DLL.
//---------------------------------------------------------------------------
-
int NSNetwork::LoadWeights(LPCSTR pathName)
{
      if (!IsInitialized())
            return -1;
      return m_NSLoadWeights(m_pNetworkInstance,pathName);
}

//---------------------------------------------------------------------------
-
// ÊËÀÑÑ: NSNetwork               ÔÓÍÊÖÈß: RandomizeWeights()
//---------------------------------------------------------------------------
-
// Ñìåøàéòå âåñà â ñåòè.
// Âîçâðàòû 0 íà óñïåõå, -1, åñëè íå èíèöèàëèçèðîâàíî, èëè êîä îøèáêè èç
DLL.
//---------------------------------------------------------------------------
-
int NSNetwork::RandomizeWeights()
{
      if (!IsInitialized())
            return -1;
      return m_NSRandomizeWeights(m_pNetworkInstance);
}

//---------------------------------------------------------------------------
-
// CLASS: NSNetwork               FUNCTION: ResetNetwork()
//---------------------------------------------------------------------------
-
// Reset the network.                                                  //---
```

```
//-----------------------------------------------------------------------
// ÊËÀÑÑ: NSNetwork               ÔÓÍÊÖÈß: SaveWeights(LPCSTR)
//-----------------------------------------------------------------------
-
// Ñîõðàíèòå ôàéë âåñîâ.
// Âîçâðàùû 0 íà óñïåõå, -1, åñëè íå èíèöèàëèçèðîâàíî, èëè êîä îøèáêè èç
DLL.
//-----------------------------------------------------------------------
-
int NSNetwork::SaveWeights(LPCSTR pathName)
{
      if (!IsInitialized())
            return -1;
      return m_NSSaveWeights(m_pNetworkInstance,pathName);
}

//=======================================================================
=
//=======================================================================
=
// CLASS: NSLearningNetwork
//=======================================================================
=
//=======================================================================
=

//-----------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: <constructor> (LPCSTR)
//-----------------------------------------------------------------------
-
NSLearningNetwork::NSLearningNetwork(LPCSTR dllPathName):
      NSNetwork(dllPathName)
{
      //-------------------------------------------------------------------
-------
      // If the DLL load was successful, get the addresses of the learning
DLL functions.
      //-------------------------------------------------------------------
-------
      if (IsLoaded())
      {
            //---------------------------------------------------------------
------------
            // Load functions unique to learning DLLs.
            //---------------------------------------------------------------
------------
            m_NSTrain = (NSTrain)GetProcAddress(m_hDLL, "train");
            m_NSGetBestCost = (NSGetBestCost)GetProcAddress(m_hDLL,
"getBestCost");
            m_NSSetBestCost = (NSSetBestCost)GetProcAddress(m_hDLL,
"setBestCost");
            m_NSGetBestWeightsPathName =
(NSGetBestWeightsPathName)GetProcAddress(m_hDLL, "getBestWeightsPathName");
            m_NSSetBestWeightsPathName =
(NSSetBestWeightsPathName)GetProcAddress(m_hDLL, "setBestWeightsPathName");
            m_NSGetSaveBestWeightsEnabled =
(NSGetSaveBestWeightsEnabled)GetProcAddress(m_hDLL,
"getSaveBestWeightsEnabled");
            m_NSSetSaveBestWeightsEnabled =
(NSSetSaveBestWeightsEnabled)GetProcAddress(m_hDLL,
"setSaveBestWeightsEnabled");
```

```
            m_NSGetCrossValidationEnabled =
(NSGetCrossValidationEnabled)GetProcAddress(m_hDLL,
"getCrossValidationEnabled");
            m_NSSetCrossValidationEnabled =
(NSSetCrossValidationEnabled)GetProcAddress(m_hDLL,
"setCrossValidationEnabled");
            m_NSGetAutoComputeInputNormCoeff =
(NSGetAutoComputeInputNormCoeff)GetProcAddress(m_hDLL,
"getAutoComputeInputNormCoeff");
            m_NSSetAutoComputeInputNormCoeff =
(NSSetAutoComputeInputNormCoeff)GetProcAddress(m_hDLL,
"setAutoComputeInputNormCoeff");
            m_NSGetAutoComputeOutputNormCoeff =
(NSGetAutoComputeOutputNormCoeff)GetProcAddress(m_hDLL,
"getAutoComputeOutputNormCoeff");
            m_NSSetAutoComputeOutputNormCoeff =
(NSSetAutoComputeOutputNormCoeff)GetProcAddress(m_hDLL,
"setAutoComputeOutputNormCoeff");
            m_NSRemoveInputNormalization =
(NSRemoveInputNormalization)GetProcAddress(m_hDLL,
"removeInputNormalization");
            m_NSRemoveOutputNormalization =
(NSRemoveOutputNormalization)GetProcAddress(m_hDLL,
"removeOutputNormalization");
            m_NSGetInputNormMin = (NSGetInputNormMin)GetProcAddress(m_hDLL,
"getInputNormMin");
            m_NSSetInputNormMin = (NSSetInputNormMin)GetProcAddress(m_hDLL,
"setInputNormMin");
            m_NSGetInputNormMax = (NSGetInputNormMax)GetProcAddress(m_hDLL,
"getInputNormMax");
            m_NSSetInputNormMax = (NSSetInputNormMax)GetProcAddress(m_hDLL,
"setInputNormMax");
            m_NSGetOutputNormMin = (NSGetOutputNormMin)GetProcAddress(m_hDLL,
"getOutputNormMin");
            m_NSSetOutputNormMin = (NSSetOutputNormMin)GetProcAddress(m_hDLL,
"setOutputNormMin");
            m_NSGetOutputNormMax = (NSGetOutputNormMax)GetProcAddress(m_hDLL,
"getOutputNormMax");
            m_NSSetOutputNormMax = (NSSetOutputNormMax)GetProcAddress(m_hDLL,
"setOutputNormMax");
            m_NSGetNormalizeInputByChannel =
(NSGetNormalizeInputByChannel)GetProcAddress(m_hDLL,
"getNormalizeInputByChannel");
            m_NSSetNormalizeInputByChannel =
(NSSetNormalizeInputByChannel)GetProcAddress(m_hDLL,
"setNormalizeInputByChannel");
            m_NSGetNormalizeOutputByChannel =
(NSGetNormalizeOutputByChannel)GetProcAddress(m_hDLL,
"getNormalizeOutputByChannel");
            m_NSSetNormalizeOutputByChannel =
(NSSetNormalizeOutputByChannel)GetProcAddress(m_hDLL,
"setNormalizeOutputByChannel");
            m_NSGetCrossValidationCostData =
(NSGetCrossValidationCostData)GetProcAddress(m_hDLL,
"getCrossValidationCostData");
            m_NSGetCostData = (NSGetCostData)GetProcAddress(m_hDLL,
"getCostData");
```

```
            m_NSGetNumberOfEpochsTrained =
(NSGetNumberOfEpochsTrained)GetProcAddress(m_hDLL,
"getNumberOfEpochsTrained");
            m_NSGetEpochOfBestCost =
(NSGetEpochOfBestCost)GetProcAddress(m_hDLL, "getEpochOfBestCost");
            m_NSSeedRandom = (NSSeedRandom)GetProcAddress(m_hDLL,
"seedRandom");

            //-----------------------------------------------------------------
------------
            // Verify functions unique to learning DLLs loaded correctly.
            // Note: Do not free DLL on failure so that caller can use
IsLoaded()/IsInitialized().
            //-----------------------------------------------------------------
------------
            if (!m_NSTrain ||
                  !m_NSGetBestCost ||
                  !m_NSSetBestCost ||
                  !m_NSGetBestWeightsPathName ||
                  !m_NSSetBestWeightsPathName ||
                  !m_NSGetSaveBestWeightsEnabled ||
                  !m_NSSetSaveBestWeightsEnabled ||
                  !m_NSGetSaveBestWeightsForTraining ||
                  !m_NSSetSaveBestWeightsForTraining ||
                  !m_NSGetCrossValidationEnabled ||
                  !m_NSSetCrossValidationEnabled ||
                  !m_NSGetAutoComputeInputNormCoeff ||
                  !m_NSSetAutoComputeInputNormCoeff ||
                  !m_NSGetAutoComputeOutputNormCoeff ||
                  !m_NSSetAutoComputeOutputNormCoeff ||
                  !m_NSRemoveInputNormalization ||
                  !m_NSRemoveOutputNormalization ||
                  !m_NSGetInputNormMin ||
                  !m_NSSetInputNormMin ||
                  !m_NSGetInputNormMax ||
                  !m_NSSetInputNormMax ||
                  !m_NSGetOutputNormMin ||
                  !m_NSSetOutputNormMin ||
                  !m_NSGetOutputNormMax ||
                  !m_NSSetOutputNormMax ||
                  !m_NSGetNormalizeInputByChannel ||
                  !m_NSSetNormalizeInputByChannel ||
                  !m_NSGetNormalizeOutputByChannel ||
                  !m_NSSetNormalizeOutputByChannel ||
                  !m_NSGetCrossValidationCostData ||
                  !m_NSGetCostData ||
                  !m_NSGetNumberOfEpochsTrained ||
                  !m_NSGetEpochOfBestCost ||
                  !m_NSSeedRandom)
            {
            }
            //-----------------------------------------------------------------
------------
            // If all functions available, create an instance of the neural
network.
            //-----------------------------------------------------------------
------------
            else
            {
```

```
                m_NSCreateNetwork(m_pNetworkInstance,LEARNING);
            }
      }
}

//---------------------------------------------------------------------------
-
// ÊËÀÑÑ: NSLearningNetwork            ÔÓÍÊÖÈß: RemoveInputNormalization()
//---------------------------------------------------------------------------
-
// Èñêëþ÷èòå íîðìàëèçàöèþ íà÷àëüíûõ äàííûõ.
// Âîçâðàòû 0 íà óñïåõ, -1, åñëè íå èíèöèàëèçèðîâàíî, èëè êîä îøèáêè èç
DLL.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::RemoveInputNormalization()
{
      if (!IsInitialized())
            return -1;
      return m_NSRemoveInputNormalization(m_pNetworkInstance);
}

//---------------------------------------------------------------------------
-
// ÊËÀÑÑ: NSLearningNetwork            ÔÓÍÊÖÈß: RemoveOutputNormalization()
//---------------------------------------------------------------------------
-
// Èñêëþ÷èòå íîðìàëèçàöèþ âûõîäíûõ  äàííûõ.
// Âîçâðàòû 0 íà óñïåõ, -1, åñëè íå èíèöèàëèçèðîâàíî, èëè êîä îøèáêè èç
DLL.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::RemoveOutputNormalization()
{
      if (!IsInitialized())
            return -1;
      return m_NSRemoveOutputNormalization(m_pNetworkInstance);
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork            FUNCTION: SeedRandom(unsigned int)
//---------------------------------------------------------------------------
-
// Seed the random number generator.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::SeedRandom(unsigned int seed)
{
      if (!IsInitialized())
            return -1;
      return m_NSSeedRandom(m_pNetworkInstance, seed);
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork            FUNCTION:
Train(int,int,double*,double*)
//---------------------------------------------------------------------------
-
// Train the network without cross-validation.
```

```
      return
m_NSTrain(m_pNetworkInstance,epochs,exemplars,inputData,desiredData,0,0,0);
}

//-------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork            FUNCTION:
Train(int,int,double*,double*,int,double*,double*)
//-------------------------------------------------------------------------
-
// Train the network with cross-validation.
// NOTE: SetCrossValidationEnabled(true) must be called for cross-validation
to be activated.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//-------------------------------------------------------------------------
-
int NSLearningNetwork::Train(int epochs, int exemplars, double* inputData,
double* desiredData, int cvExemplars, double* cvInputData, double*
cvDesiredData)
{
      if (!IsInitialized())
            return -1;
      return
m_NSTrain(m_pNetworkInstance,epochs,exemplars,inputData,desiredData,cvExempl
ars,cvInputData,cvDesiredData);
}

//-------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetAutoComputeInputNormCoeff()
//-------------------------------------------------------------------------
-
// Return whether or not the input normalization coefficients will
// automatically be calculated.
//-------------------------------------------------------------------------
-
bool NSLearningNetwork::GetAutoComputeInputNormCoeff()
{
      if (!IsInitialized())
            return false;
      bool autoComputeInputNormCoeff;
      if
(m_NSGetAutoComputeInputNormCoeff(m_pNetworkInstance,autoComputeInputNormCoe
ff))
            return false;
      return autoComputeInputNormCoeff;
}

//-------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetAutoComputeOutputNormCoeff()
//-------------------------------------------------------------------------
-
// Return whether or not the output normalization coefficients will
// automatically be calculated.
//-------------------------------------------------------------------------
-
bool NSLearningNetwork::GetAutoComputeOutputNormCoeff()
{
      if (!IsInitialized())
            return false;
      bool autoComputeOutputNormCoeff;
```

```
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetBestCost()
//---------------------------------------------------------------------------
-
// Return the best cost or -1 on failure.
//---------------------------------------------------------------------------
-
double NSLearningNetwork::GetBestCost()
{
      double bestCost;
      if (!IsInitialized())
            return -1;
      if (m_NSGetBestCost(m_pNetworkInstance,bestCost))
            return -1;
      return bestCost;
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetBestWeightsPathName()
//---------------------------------------------------------------------------
-
// Return the best weights path name or an empty string on failure.
//---------------------------------------------------------------------------
-
CString NSLearningNetwork::GetBestWeightsPathName()
{
      char buffer[1024];
      int bufferUsed;
      if (!IsInitialized())
            return "";
      if
(m_NSGetBestWeightsPathName(m_pNetworkInstance,buffer,sizeof(buffer),bufferU
sed))
            return "";
      return buffer;
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetCostData(double *)
//---------------------------------------------------------------------------
-
// Fills the costData array with the training learning curve (an array of
cost
// data for the training dataset). You must allocate the memory for the cost
// data before passing its pointer to this function. Use
GetNumberOfEpochsTrained
// to determine the necessary size of the array.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::GetCostData(double *costData)
{
      if (!IsInitialized())
            return -1;
      return m_NSGetCostData(m_pNetworkInstance, costData);
}
                                                                        //
```

```
array of cost data for the cross validation dataset). You must allocate
// the memory for the cost data before passing its pointer to this function.
// Use GetNumberOfEpochsTrained to determine the necessary size of the
array.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::GetCrossValidationCostData(double *CVCostData)
{
      if (!IsInitialized())
            return -1;
      return m_NSGetCrossValidationCostData(m_pNetworkInstance, CVCostData);
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetCrossValidationEnabled()
//---------------------------------------------------------------------------
-
// Return whether or not cross validation will be used during training.
//---------------------------------------------------------------------------
-
bool NSLearningNetwork::GetCrossValidationEnabled()
{
      if (!IsInitialized())
            return false;
      bool crossValidationEnabled;
      if
(m_NSGetCrossValidationEnabled(m_pNetworkInstance,crossValidationEnabled))
            return false;
      return crossValidationEnabled;
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetEpochOfBestCost()
//---------------------------------------------------------------------------
-
// Return the epoch during which the best cost was obtained.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::GetEpochOfBestCost()
{
      if (!IsInitialized())
            return -1;
      int epochOfBestCost;
      m_NSGetEpochOfBestCost(m_pNetworkInstance, epochOfBestCost);
      return epochOfBestCost;
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetInputNormMax()
//---------------------------------------------------------------------------
-
// Returns the upper bound for the input normalization.
//---------------------------------------------------------------------------
-
double NSLearningNetwork::GetInputNormMax()
{
      if (!IsInitialized())
            return -1;
```

```
//-----------------------------------------------------------------------
-
// Returns the lower bound for the input normalization.
//-----------------------------------------------------------------------
-
double NSLearningNetwork::GetInputNormMin()
{
      if (!IsInitialized())
            return -1;
      double inputNormMin;
      m_NSGetInputNormMin(m_pNetworkInstance, inputNormMin);
      return inputNormMin;
}

//-----------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetNormalizeInputByChannel()
//-----------------------------------------------------------------------
-
// Return whether or not the input normalization coefficients will
// calculated by channel (for each column of data individually) or across
the
// entire input dataset.
//-----------------------------------------------------------------------
-
bool NSLearningNetwork::GetNormalizeInputByChannel()
{
      if (!IsInitialized())
            return false;
      bool normalizeInputByChannel;
      if (m_NSGetNormalizeInputByChannel(m_pNetworkInstance,
normalizeInputByChannel))
            return false;
      return normalizeInputByChannel;
}

//-----------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetNormalizeOutputByChannel()
//-----------------------------------------------------------------------
-
// Return whether or not the output normalization coefficients will
// calculated by channel (for each column of data individually) or across
the
// entire output dataset.
//-----------------------------------------------------------------------
-
bool NSLearningNetwork::GetNormalizeOutputByChannel()
{
      if (!IsInitialized())
            return false;
      bool normalizeOutputByChannel;
      if (m_NSGetNormalizeOutputByChannel(m_pNetworkInstance,
normalizeOutputByChannel))
            return false;
      return normalizeOutputByChannel;
}

//-----------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetNumberOfEpochsTrained()
//-----------------------------------------------------------------------
-                                                                    int
```

```
numberOfEpochsTrained;
      m_NSGetNumberOfEpochsTrained(m_pNetworkInstance,
numberOfEpochsTrained);
      return numberOfEpochsTrained;
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetOutputNormMax()
//---------------------------------------------------------------------------
-
// Returns the upper bound for the output normalization.
//---------------------------------------------------------------------------
-
double NSLearningNetwork::GetOutputNormMax()
{
      if (!IsInitialized())
            return -1;
      double outputNormMax;
      m_NSGetOutputNormMax(m_pNetworkInstance, outputNormMax);
      return outputNormMax;
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetOutputNormMin()
//---------------------------------------------------------------------------
-
// Returns the upper bound for the output normalization.
//---------------------------------------------------------------------------
-
double NSLearningNetwork::GetOutputNormMin()
{
      if (!IsInitialized())
            return -1;
      double outputNormMin;
      m_NSGetOutputNormMin(m_pNetworkInstance, outputNormMin);
      return outputNormMin;
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetSaveBestWeightsEnabled()
//---------------------------------------------------------------------------
-
// Return whether or not the best weights will be saved during training.
//---------------------------------------------------------------------------
-
bool NSLearningNetwork::GetSaveBestWeightsEnabled()
{
      bool saveBestWeightsEnabled;
      if (!IsInitialized())
            return false;
      if
(m_NSGetSaveBestWeightsEnabled(m_pNetworkInstance,saveBestWeightsEnabled))
            return false;
      return saveBestWeightsEnabled;
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: GetSaveBestWeightsForTraining()
```

```
            return false;
      if
(m_NSGetSaveBestWeightsForTraining(m_pNetworkInstance,saveBestWeightsForTrai
ning))
            return false;
      return saveBestWeightsForTraining;
}

//-------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION:
SetAutoComputeInputNormCoeff(bool)
//-------------------------------------------------------------------------
-
// Turn the auto-computation of input normalization coefficients on or off.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//-------------------------------------------------------------------------
-
int NSLearningNetwork::SetAutoComputeInputNormCoeff(bool
autoComputeInputNormCoeff)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetAutoComputeInputNormCoeff(m_pNetworkInstance,
autoComputeInputNormCoeff);
}

//-------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION:
SetAutoComputeOutputNormCoeff(bool)
//-------------------------------------------------------------------------
-
// Turn the auto-computation of output normalization coefficients on or off.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//-------------------------------------------------------------------------
-
int NSLearningNetwork::SetAutoComputeOutputNormCoeff(bool
autoComputeOutputNormCoeff)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetAutoComputeOutputNormCoeff(m_pNetworkInstance,
autoComputeOutputNormCoeff);
}

//-------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: SetBestCost(double)
//-------------------------------------------------------------------------
-
// Set the best cost.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//-------------------------------------------------------------------------
-
int NSLearningNetwork::SetBestCost(double bestCost)
{
      if (!IsInitialized())
            return -1;
      return m NSSetBestCost(m pNetworkInstance,bestCost);                    //
```

```
Returns 0 on success, -1 if not initialized, or an error code from the DLL.
//-----------------------------------------------------------------------------
-
int NSLearningNetwork::SetBestWeightsPathName(LPCSTR pathName)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetBestWeightsPathName(m_pNetworkInstance,pathName);
}

//-----------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: SetCrossValidationEnabled(bool)
//-----------------------------------------------------------------------------
-
// Turn the cross-validation on or off.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//-----------------------------------------------------------------------------
-
int NSLearningNetwork::SetCrossValidationEnabled(bool
crossValidationEnabled)
{
      if (!IsInitialized())
            return -1;
      return
m_NSSetCrossValidationEnabled(m_pNetworkInstance,crossValidationEnabled);
}

//-----------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: SetInputNormMax(double)
//-----------------------------------------------------------------------------
-
// Sets the upper bound for the input normalization.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//-----------------------------------------------------------------------------
-
int NSLearningNetwork::SetInputNormMax(double inputNormMax)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetInputNormMax(m_pNetworkInstance, inputNormMax);
}

//-----------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: SetInputNormMin(double)
//-----------------------------------------------------------------------------
-
// Sets the lower bound for the input normalization.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//-----------------------------------------------------------------------------
-
int NSLearningNetwork::SetInputNormMin(double inputNormMin)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetInputNormMin(m_pNetworkInstance, inputNormMin);
}
```

```
int NSLearningNetwork::SetNormalizeInputByChannel(bool
normalizeInputByChannel)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetNormalizeInputByChannel(m_pNetworkInstance,
normalizeInputByChannel);
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION:
SetNormalizeOutputByChannel(bool)
//---------------------------------------------------------------------------
-
// Turn by-channel normalization of the output data on or off.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::SetNormalizeOutputByChannel(bool
normalizeOutputByChannel)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetNormalizeOutputByChannel(m_pNetworkInstance,
normalizeOutputByChannel);
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: SetOutputNormMax(double)
//---------------------------------------------------------------------------
-
// Sets the upper bound for the output normalization.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::SetOutputNormMax(double outputNormMax)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetOutputNormMax(m_pNetworkInstance, outputNormMax);
}

//---------------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION: SetOutputNormMin(double)
//---------------------------------------------------------------------------
-
// Sets the lower bound for the output normalization.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//---------------------------------------------------------------------------
-
int NSLearningNetwork::SetOutputNormMin(double outputNormMin)
{
      if (!IsInitialized())
            return -1;
      return m_NSSetOutputNormMin(m_pNetworkInstance, outputNormMin);
}
                                                                        //---
```

```
//---------------------------------------------------------------------
int NSLearningNetwork::SetSaveBestWeightsEnabled(bool
saveBestWeightsEnabled)
{
      if (!IsInitialized())
            return -1;
      return
m_NSSetSaveBestWeightsEnabled(m_pNetworkInstance,saveBestWeightsEnabled);
}

//---------------------------------------------------------------------
-
// CLASS: NSLearningNetwork       FUNCTION:
SetSaveBestWeightsForTraining(bool)
//---------------------------------------------------------------------
-
// Turn the saving of the best weights for training on or off.
// Returns 0 on success, -1 if not initialized, or an error code from the
DLL.
//---------------------------------------------------------------------
-
int NSLearningNetwork::SetSaveBestWeightsForTraining(bool
saveBestWeightsForTraining)
{
      if (!IsInitialized())
            return -1;
      return
m_NSSetSaveBestWeightsForTraining(m_pNetworkInstance,saveBestWeightsForTrain
ing);
}

//=====================================================================
=
//=====================================================================
=
// CLASS: NSRecallNetwork
//=====================================================================
=
//=====================================================================
=

//---------------------------------------------------------------------
-
// CLASS: NSRecallNetwork         FUNCTION: <constructor> (LPCSTR)
//---------------------------------------------------------------------
-
NSRecallNetwork::NSRecallNetwork(LPCSTR dllPathName):
      NSNetwork(dllPathName)
{
      //-----------------------------------------------------------------
-------
      // If DLL loaded successfully, create an instance of the neural
network.
      //-----------------------------------------------------------------
-------
      if (IsLoaded())
      {
            m_NSCreateNetwork(m_pNetworkInstance,RECALL);
      }
}
//=====================================================================
=
// Custom Solution Wizard - Visual C++ Shell - Sample Functions
```

```
//    It uses the training data taken from your original breadboard. This
data
//  was saved to two files: InputData.txt and DesiredData.txt during the DLL
//  generation.
//=============================================================================
=

//=============================================================================
=
// Includes
//=============================================================================
=
#include "stdafx.h"
#include <string.h>
#include "VCPPShell.h"
#include "NSNetwork.h"
#include "Globals.h"
#include ".\vcppshell.h"
#include <stdio.h>
#include "odbcinst.h"
#include <AFXDB.H>
#include <windows.h>
#include <windowsx.h>
//=============================================================================
=
// Debug handling generated by MFC AppWizard.
//=============================================================================
=
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//=============================================================================
=
//=============================================================================
=
// CLASS: CVCPPShellApp
//=============================================================================
=
//=============================================================================
=

//=============================================================================
=
// Global declaration of application generated by MFC AppWizard.
//=============================================================================
=
CVCPPShellApp theApp;

//=============================================================================
=
// CLASS: CVCPPShellApp                MESSAGE MAP
//-----------------------------------------------------------------------------
-
// Note: This mapping was built by the MFC AppWizard.
//=============================================================================
=
BEGIN_MESSAGE_MAP(CVCPPShellApp, CWinApp)
      //{{AFX_MSG_MAP(CVCPPShellApp)
            // NOTE - the ClassWizard will add and remove mapping macros
here.                                                                    }
```

```
//=============================================================================
=
// CLASS: CVCPPShellApp                FUNCTION: InitInstance()
//-----------------------------------------------------------------------------
-
// Note: This function was built by the MFC AppWizard.
//=============================================================================
=
BOOL CVCPPShellApp::InitInstance()
{
      // Standard initialization
      // If you are not using these features and wish to reduce the size
      //  of your final executable, you should remove from the following
      //  the specific initialization routines you do not need.

      CVCPPShellDlg dlg;
      m_pMainWnd = &dlg;
      int nResponse = dlg.DoModal();
      if (nResponse == IDOK)
      {
            //  Place code here to handle when the dialog is
            //  dismissed with OK
      }
      else if (nResponse == IDCANCEL)
      {
            //  Place code here to handle when the dialog is
            //  dismissed with Cancel
      }

      // Since the dialog has been closed, return FALSE so that we exit the
      //  application, rather than start the application's message pump.
      return FALSE;
}

      int g=0;
//=============================================================================
=
//=============================================================================
=
// CLASS: CVCPPShellDlg
//=============================================================================
=
//=============================================================================
=

//=============================================================================
=
// CLASS: CVCPPShellDlg                FUNCTION: <constructor>(CWnd*)
//-----------------------------------------------------------------------------
-
// Note: This function was built by the MFC AppWizard.
//=============================================================================
=
CVCPPShellDlg::CVCPPShellDlg(CWnd* pParent /*=NULL*/)
      : CDialog(CVCPPShellDlg::IDD, pParent)
{
      //{{AFX_DATA_INIT(CVCPPShellDlg)

      //}}AFX_DATA_INIT
      // Note that LoadIcon does not require a subsequent DestroyIcon in
Win32
      m hIcon = AfxGetApp()->LoadIcon(IDR MAINFRAME);
```

```
// Note: This function was built by the MFC AppWizard.
//=============================================================================
=
void CVCPPShellDlg::DoDataExchange(CDataExchange* pDX)
{
      CDialog::DoDataExchange(pDX);
      //{{AFX_DATA_MAP(CVCPPShellDlg)
      DDX_Text(pDX, IDC_TRAIN_NETWORK_REPORT, m_TrainNetwork_Report);
      //}}AFX_DATA_MAP

}


//=============================================================================
=
// CLASS: CVCPPShellDlg                FUNCTION: FillArrayFromFile(LPCTSTR,
int, int)
//=============================================================================
=
double *CVCPPShellDlg::FillArrayFromFile(LPCTSTR fName, int numRows, int
numCols)
{
      CStdioFile fl;
      LPTSTR tmpBuffer = new char [250];

      char *token;                      //pointer to string tokens to extract
from file strings
      char *delim = " ,\t\n";           //delimiter to use for string
tokenizer

      //create array to hold double values extracted from file
      long numCells = numRows * numCols;
      double *retArray = (double *)calloc (numCells, sizeof (double));
      int curIndx = 0;
      CButton *Vvod = (CButton *) GetDlgItem (IDC_RESET_NETWORK2);

      if (fl.Open(fName, CFile::modeRead))
      {

            //read file until end is reached
            while (((fl.ReadString(tmpBuffer, 250)) != NULL) && ((curIndx <
numCells)))
            {
                  //tokenize string to extract double values as strings
                  token = strtok (tmpBuffer, delim);

                  while (token != NULL)
                  {
                        //convert token to double and store in array
                        retArray[curIndx] = (double) atof (token);
                        curIndx++;
                        //tokenize string to extract double values as strings
                        token = strtok (NULL, delim);

                  } //end token while


            } //end fl.Open while

            fl.Close();
      } //end if
                                                                        //e
```

```
nd if


      free (tmpBuffer);
      return retArray;
} //end FillArrayFromFile


//=============================================================================
=
// CLASS: CVCPPShellDlg                MESSAGE MAP
//-----------------------------------------------------------------------------
-
// Note: This mapping was built by the MFC AppWizard.
//=============================================================================
=
BEGIN_MESSAGE_MAP(CVCPPShellDlg, CDialog)
      //{{AFX_MSG_MAP(CVCPPShellDlg)
      ON_WM_PAINT()
      ON_WM_QUERYDRAGICON()
      ON_BN_CLICKED(IDC_GET_NETWORK_OUTPUT, OnGetResponse)
      ON_BN_CLICKED(IDC_TRAIN_NETWORK, OnTrainNetwork)
      ON_BN_CLICKED(IDC_BUTTON3, OnButton3)
      ON_BN_CLICKED(IDC_RADIO1, OnRadio1)
      //}}AFX_MSG_MAP
      ON_EN_CHANGE(IDC_GET_NETWORK_OUTPUT_REPORT,
OnEnChangeGetNetworkOutputReport)
      ON_EN_CHANGE(IDC_TRAIN_NETWORK_REPORT, OnEnChangeTrainNetworkReport)
      ON_BN_CLICKED(IDC_BUTTON1, OnBnClickedButton1)
      ON_BN_CLICKED(IDC_BUTTON2, OnBnClickedButton2)
      ON_BN_CLICKED(IDC_BUTTON3,OnButton3)
      ON_BN_CLICKED(IDC_RESET_NETWORK, OnBnClickedResetNetwork)
      ON_BN_CLICKED(IDC_RESET_NETWORK2, OnBnClickedResetNetwork2)

END_MESSAGE_MAP()


//=============================================================================
=
// CLASS: CVCPPShellDlg                FUNCTION: OnInitDialog()
//-----------------------------------------------------------------------------
-
// Note: This function was built by the MFC AppWizard.
//=============================================================================
=
BOOL CVCPPShellDlg::OnInitDialog()
{

      CDialog::OnInitDialog();

      // Set the icon for this dialog.  The framework does this
automatically
      // when the application's main window is not a dialog
      SetIcon(m_hIcon, TRUE);                  // Set big icon
      SetIcon(m_hIcon, FALSE);          // Set small icon

      //dispaly initial message in Get Network Output Report edit box
      CEdit *pGetNetworkOutputReportEditBox = (CEdit *) GetDlgItem
(IDC_GET_NETWORK_OUTPUT_REPORT);
      pGetNetworkOutputReportEditBox->SetWindowText(_T("Not tested."));

      //disables training if the DLL only supports a recall network
      if (RECALL ONLY NETWORK)
```

```
                CWnd *pTrainNetworkButton = (CWnd *) GetDlgItem
(IDC_TRAIN_NETWORK);
                pTrainNetworkButton->EnableWindow(FALSE);

                //disable Reset Network Before Training check box
                CButton *pResetNetworkCheckBox = (CButton *) GetDlgItem
(IDC_RESET_NETWORK);
                pResetNetworkCheckBox->EnableWindow(FALSE);

                //dispaly initial message in Train Network Report edit box
                CEdit *pTrainNetworkReportEditBox = (CEdit *) GetDlgItem
(IDC_TRAIN_NETWORK_REPORT);
                pTrainNetworkReportEditBox->SetWindowText(_T("Only Recall
DLL Supported."));

                m_TrainNetwork_Report = _T("Recall Network Only.");
        }
        else
        {
                //dispaly initial message in Train Network Report edit box
                CEdit *pTrainNetworkReportEditBox = (CEdit *) GetDlgItem
(IDC_TRAIN_NETWORK_REPORT);
                pTrainNetworkReportEditBox->SetWindowText(_T("Not
tested."));

                m_TrainNetwork_Report = _T("Not tested.");
        }

        return TRUE;  // return TRUE  unless you set the focus to a control
}

//=============================================================================
=
// CLASS: CVCPPShellDlg                 FUNCTION: OnPaint()
//-----------------------------------------------------------------------------
-
// Note: This function was built by the MFC AppWizard.
//-----------------------------------------------------------------------------
-
// If you add a minimize button to your dialog, you will need the code below
//  to draw the icon.  For MFC applications using the document/view model,
//  this is automatically done for you by the framework.
//=============================================================================
=
void CVCPPShellDlg::OnPaint()
{
        if (IsIconic())
        {
                CPaintDC dc(this); // device context for painting

                SendMessage(WM_ICONERASEBKGND, (WPARAM) dc.GetSafeHdc(), 0);

                // Center icon in client rectangle
                int cxIcon = GetSystemMetrics(SM_CXICON);
                int cyIcon = GetSystemMetrics(SM_CYICON);
                CRect rect;
                GetClientRect(&rect);
                int x = (rect.Width() - cxIcon + 1) / 2;
                int y = (rect.Height() - cyIcon + 1) / 2;

                // Draw the icon
```

```
		dc.DrawIcon(x, y, m_hIcon);
	}
	else
	{
		CDialog::OnPaint();
	}
}

//=============================================================================
=
// CLASS: CVCPPShellDlg                FUNCTION: OnQueryDragIcon()
//-----------------------------------------------------------------------------
-
// Note: This function was built by the MFC AppWizard.
//-----------------------------------------------------------------------------
-
// The system calls this to obtain the cursor to display while the user
drags
//  the minimized window.
//=============================================================================
=
HCURSOR CVCPPShellDlg::OnQueryDragIcon()
{
	return (HCURSOR) m_hIcon;
}

//=============================================================================
=
// CLASS: CVCPPShellDlg                FUNCTION: OnGetResponse()
//-----------------------------------------------------------------------------
-
//  This function illustrates how the DLL generated by the Custom Solution
//  Wizard can be used for getting the network response (output).
//  Step by step instructions are given on how to use the generated DLL
//  for this purpose.  Read through comments carefully performing all of the
//  REQUIRED ACTIONs as you get to them.
//=============================================================================
=
void CVCPPShellDlg::OnBnClickedButton1()
{
	//
		{
  CDatabase database;
  CString sDriver = "MICROSOFT EXCEL DRIVER (*.XLS)"; // название драйвера в
ODBC-менеджере
  CString sExcelFile = "Out.xls";                // имя и путь для
создаваемого файла
  CString sSql;
  double a;

  TRY
  {
    // Форматируем строку для доступа минуя DSN


sSql.Format("DRIVER={%s};DSN='';FIRSTROWHASNAMES=1;READONLY=FALSE;CREATE_DB=
\"%s\";DBQ=%s",
                sDriver, sExcelFile, sExcelFile);

    // Создаём базу данных (всмысле таблицу Excel)
    if( database.OpenEx(sSql,CDatabase::noOdbcDialog) )
    {
      // Создаём структуру таблицы
```

```
      sSql = "INSERT INTO demo_table (Field1,Field2) VALUES (1,2)";
      database.ExecuteSQL(sSql);
      a=2.6;
      sSql = "INSERT INTO demo_table (Field1,Field2) VALUES (a,30)";
      database.ExecuteSQL(sSql);

      sSql = "INSERT INTO demo_table (Field1,Field2) VALUES (s1[1],s2[1])";
      database.ExecuteSQL(sSql);
    }

    // Закрываем базу данных
    database.Close();
  }
  CATCH_ALL(e)
  {
    TRACE1("Driver not installed: %s",sDriver);
  }
  END_CATCH_ALL;
}

////////////////////////////// ///////////////// ////////////////////




      }

void CVCPPShellDlg::OnBnClickedButton2()
{
      STARTUPINFO si = { sizeof(si) };
      PROCESS_INFORMATION pi;
      CreateProcess(NULL, "files/graphik.exe", NULL, NULL, TRUE,
NULL, NULL, "files/", &si, &pi);

}


void CVCPPShellDlg::OnGetResponse()
{
      //-------------------------------------------------------------------
-------
      // Шаг 1: Образец(загрузить DLL) Создать используя класс
NSRecallNetwork. (кстати там всего одна функция...)
      //-------------------------------------------------------------------
-------
    // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ: DLL_PATH_NAME определен в Globals.h
      //-------------------------------------------------------------------
-------
      // Следование за этим вызовом, IsInitialized() должен вернуть истину.
      // Если IsInitialized() - это ложь, чек, чтобы убедиться, что путь в
DLL правилен.
      //-------------------------------------------------------------------
-------
    // Вы могли бы создать объект NSLearningNetwork вместо этого, если вы
имеете
    // Горизонтальные Кудесника Таможенного Решения разработчики и
использовал это наряду с
      // Макет NeuroSolutions, способный к изучению, чтобы генерировать DLL.
```

```
      // Как NSRecallNetwork, так и действия отозвания поддержки
NSLearningNetwork.
      //-------------------------------------------------------------------
-------
      NSRecallNetwork nn(DLL_PATH_NAME);

      if (nn.IsInitialized())
      {
            //-------------------------------------------------------------
------------
            // Шаг 2: Получаем размер из DLL (количество входов и выходов)
            //-------------------------------------------------------------
------------
            int inputs;
            int outputs;
            nn.GetInputOutputInfo(inputs,outputs);

            //-------------------------------------------------------------
------------
            // Step 3: инициализировать нейронную сеть weights.
            //-------------------------------------------------------------
------------
            // IMPLEMENTATION NOTES: The BEST_WEIGHTS_PATH_NAME is defined in
Globals.h.
            // The initial best weights file is an exact copy of the weights
file that was
            // saved when the DLL was generated. However, the best weights
file will change
            // with each run of the TrainNetwork function if the network is
reset before
            // training.
            //-------------------------------------------------------------
------------
            nn.LoadWeights(BEST_WEIGHTS_PATH_NAME);

            //-------------------------------------------------------------
------------
            // Step 4: Определите начальные данные.
            //-------------------------------------------------------------
------------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ: Следующее загружает учебные начальные
данные из вашей начальной буквы
            // конфигурация сети в массив inputData.
            // Отметьте, что число входов в начальных данных должно
соответствовать числу входов
            // ожидаемый генерируемым DLL.  Также, может быть какое-нибудь
число образцов в
            // начальные данные, пока число образцов в начальных данных
соответствует числу
            // образцов в желаемых данных. В данном случае, в
NUMBER_OF_EXEMPLARS определяется
            // Globals.h.
            //-------------------------------------------------------------
------------

            double *inputData = FillArrayFromFile(INPUT_FILE_PATH_NAME,
inputs, NUMBER_OF_EXEMPLARS);
```

```
            //-----------------------------------------------------------------
------------
            // Step 5: Объявите хранение для ответа.
            //-----------------------------------------------------------------
------------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ : Измерения массива outputData
установлены
            // чтобы соответствовать ожидаемому размеру выходных данных.
            //-----------------------------------------------------------------
------------
            // Примечание: Первое измерение массива outputData сортируется по
величине, согласовывая
            // к числу выходов в DLL. Второе измерение outputData
            // массив сортируется по величине согласно числу образцов в вашем
наборе данных как
            // определенный в Globals.h.
            //-----------------------------------------------------------------
------------
            double *outputData = (double *) calloc (1 * NUMBER_OF_EXEMPLARS,
sizeof (double));

            //-----------------------------------------------------------------
------------
            // Step 6: Доберитесь ответ сети до начальных данных.
            //-----------------------------------------------------------------
------------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ: NUMBER_OF_EXEMPLARS определен в
Globals.h.
            //-----------------------------------------------------------------
------------

            nn.GetResponse(NUMBER_OF_EXEMPLARS, inputData, outputData);


            //шаг 1-6 были ??????? нейронной сети...
            //-----------------------------------------------------------------
------------
            // Step 7: Проверьте результаты. (Необязательный)
            //-----------------------------------------------------------------
------------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ: Выходы результаты к блоку
редактирования Отчета Выхода Сети Получения
            //-----------------------------------------------------------------
------------
            // Подготовьте блок редактирования для выхода -> очищают его
содержимое.
            CEdit *pGetNetworkOutputReportEditBox = (CEdit *) GetDlgItem
(IDC_GET_NETWORK_OUTPUT_REPORT);
            pGetNetworkOutputReportEditBox->SetWindowText(_T(""));

            CString outStr = "Input \t\t";

            // Add labels to top of edit box

                  outStr += "Output";
                        outStr += "\r\n";

            // Show outputs for all up to 1000 exemplars
```

```
            int maxExemplars = (NUMBER_OF_EXEMPLARS >
1000)?1000:NUMBER_OF_EXEMPLARS;

            for (int j = 0; j < g; j++) //если  maxExemplars=2 то вычисляется
только 1-е и 2-е значение
            {
                  CString jAsStr;
                  jAsStr.Format("%9f\t", inputData[j]); // заполняет
значениями от 1 до maxExemplars 1-й столбец
                  outStr += jAsStr;

                  for (int k = 0; k < 1; k++)
                  {
                        CString resultAsString;
                        resultAsString.Format("%9f", outputData[(j * outputs)
+ k]);
                        outStr += resultAsString;
                  } //end for j

                  outStr += "\r\n";
            } //end for i

            pGetNetworkOutputReportEditBox->SetWindowText(_T(outStr));
            CButton *Vvod = (CButton *) GetDlgItem (IDC_RESET_NETWORK3);
            if (Vvod->GetCheck() == 1)
            {
                  CString V;
      {
      // создание стандартной панели выбора файла SaveAs
      CFileDialog DlgSaveAs(FALSE,(LPCSTR)"txt",NULL,
            OFN_HIDEREADONLY|OFN_OVERWRITEPROMPT,
            (LPCSTR)" Text Files (*.txt) |*.txt||");

      // отображение стандартной панели выбора файла SaveAs
      if(DlgSaveAs.DoModal()==IDOK)
      {
            // создание объекта и открытие файла для записи
            CStdioFile File(DlgSaveAs.GetPathName(),
            CFile::modeCreate|CFile::modeWrite|CFile::typeBinary);

            // запись в файл строки
            for (int i=0; i<61; i++)
            {
            V.Format("%9f\r\n", outputData[i]);
            File.WriteString((LPCTSTR)V);
                        }
      }UpdateData(FALSE);
      }
            }

            //-----------------------------------------------------------------
------------
            // Perform some memory clean-up
            //-----------------------------------------------------------------
------------
            free (inputData);
            free (outputData);
      remove( "files/InputData.txt" );
      remove( "files/DesiredData.txt" );
```

```
	}
	//-------------------------------------------------------------------------
-------
	// If initialization failed, report any errors here.
	//-------------------------------------------------------------------------
-------
	else
	{
		//prepare edit box for output -> reset its contents
		CEdit *pGetNetworkOutputReportEditBox = (CEdit *) GetDlgItem
(IDC_GET_NETWORK_OUTPUT_REPORT);
		pGetNetworkOutputReportEditBox->SetWindowText(_T("DLL Load
Failed."));
	}

	//-------------------------------------------------------------------------
-------
	// Update the dialog.
	//-------------------------------------------------------------------------
-------
	UpdateData(FALSE);
}

//===========================================================================
=
// CLASS: CVCPPShellDlg                FUNCTION: OnTrainNetwork()
//---------------------------------------------------------------------------
-
//  Функция обучает обучающую сеть DLL и сохраняет лучшие веса
//    к указанному файлу для более позднего использования.
//  Шаг за шагом инструкции предоставлены на том, как использовать
генерируемый DLL
//  для этой цели.  Чтение через комментарии тщательно. ПРИМЕЧАНИЯ
ВЫПОЛНЕНИЯ
//  подсветите параметры, что вы, возможно, желаете измениться в будущем.
//---------------------------------------------------------------------------
-
//    ПРИМЕЧАНИЕ: Эта подпрограмма только applys потребителям горизонтальных
Разработчиков
//    Кудесник Таможенного Решения.  Уровень Разработчиков позволяет
потребителя для генерации
//    изучение Dlls, тогда как все другие уровни могут только генерировать
отозвание Dlls.
//===========================================================================
=
void CVCPPShellDlg::OnTrainNetwork()
{

	//-------------------------------------------------------------------------
-------
	// Update Train Network Report edit box to indicate that the network
is training.
	//-------------------------------------------------------------------------
-------
	CEdit *pTrainNetworkReportEditBox = (CEdit *) GetDlgItem
(IDC_TRAIN_NETWORK_REPORT);
	pTrainNetworkReportEditBox->SetSel(0, -1, FALSE);
	pTrainNetworkReportEditBox->ReplaceSel("Training...", FALSE);

	CDatabase database;
	CString sSql;
	CString sItem1, sItem2;
	CString sDriver, *a;
```

```
      CString sFile = "files/Input.xls";           // the file name. Could
also be something like C:\\Sheets\\WhatDoIKnow.xls
      CString VV,VVV;

      // Clear the contents of the listbox
//    m_ctrlList.ResetContent();

      // Retrieve the name of the Excel driver. This is
      // necessary because Microsoft tends to use language
      // specific names like "Microsoft Excel Driver (*.xls)" versus
      // "Microsoft Excel Treiber (*.xls)"
      sDriver = GetExcelDriver();
      if( sDriver.IsEmpty() )
      {
            // Blast! We didn't find that driver!
            AfxMessageBox("No Excel ODBC driver found");
            return;
      }

      // Create a pseudo DSN including the name of the Driver and the Excel
file
      // so we don't have to have an explicit DSN installed in our ODBC
admin
      sDsn.Format("ODBC;DRIVER={%s};DSN='';DBQ=%s",sDriver,sFile);

      TRY
      {
            // Open the database using the former created pseudo DSN
            database.Open(NULL,false,false,sDsn);

            // Allocate the recordset
            CRecordset recset( &database );

            // Build the SQL string
            // Remember to name a section of data in the Excel sheet using
"Insert->Names" to be
            // able to work with the data like you would with a table in a
"real" database. There
            // may be more than one table contained in a worksheet.
            sSql = "SELECT field_1, field_2 "
                        "FROM NN ";
                        //"ORDER BY field_1";

            // Execute that query (implicitly by opening the recordset)
            recset.Open(CRecordset::forwardOnly,sSql,CRecordset::readOnly);

            // Browse the result
            while( !recset.IsEOF() )
            {
                  // Read the result line
                  recset.GetFieldValue("field_1",sItem1);
                  VVV+=sItem1;
                  VVV+="\n";
                  g++;
                  recset.MoveNext();
            }

            // Close the database
            database.Close();
```

```
}
      CATCH(CDBException, e)
      {
            // A database exception occured. Pop out the details...
            AfxMessageBox("Database error: "+e->m_strError);
      }
      END_CATCH;// TODO: Add your control notification handler code here

      {
            FILE *file;
            char* file_name = "files/InputData.txt";
            char load_string[50] = "none";
            file = fopen( file_name, "w" );
            fputs( VVV, file );
            fclose( file );
            }

      g=0;VVV=' ';
      ////////////////////////////////////////////////////////////////////////
/////
      sDriver = GetExcelDriver();
      if( sDriver.IsEmpty() )
      {
                        AfxMessageBox("No Excel ODBC driver found");
            return;
      }

      sDsn.Format("ODBC;DRIVER={%s};DSN='';DBQ=%s",sDriver,sFile);

      TRY
      {

            database.Open(NULL,false,false,sDsn);
            CRecordset recset( &database );
            sSql = "SELECT field_1, field_2 "
                        "FROM NN ";
            recset.Open(CRecordset::forwardOnly,sSql,CRecordset::readOnly);
            while( !recset.IsEOF() )
            {
                  recset.GetFieldValue("field_2",sItem2);
VVV+=sItem2;
VVV+="\n";
g++;

                  recset.MoveNext();
            }
            database.Close();

      }
      CATCH(CDBException, e)
      {
            AfxMessageBox("Database error: "+e->m_strError);
      }
      END_CATCH;// TODO: Add your control notification handler code here

      {
FILE *file;
char* file_name = "files/DesiredData.txt";
char load_string[50] = "none";
```

```
file = fopen( file_name, "w" );
fputs( VVV, file );
fclose( file );
      }

      //-------------------------------------------------------------------------
      // Step 1: Create an instance of NSLearningNetwork.
      //-------------------------------------------------------------------------
    // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ: Путь DLL определен в Globals.h.
      //-------------------------------------------------------------------------
      // Следование за этим вызовом, IsInitialized() должен вернуть истину.
      // Если IsInitialized() - это ложь, одна из следующих ошибок occured:
      //          * Путь в DLL некорректен.  IsLoaded() будет ложью.
      //          * Путь указывает отозвание DLL.  IsLoaded() будет истинен.
      //-------------------------------------------------------------------------
      // Note: This step will fail if you are not using the Developers level of
      // the Custom Solution Wizard or if you generated the DLL using a recall
      // NeuroSolutions breadboard.
      //-------------------------------------------------------------------------
      NSLearningNetwork nn(DLL_PATH_NAME);

      if (nn.IsInitialized())
      {
            //-------------------------------------------------------------------------
------
            // Step 2: Получите размер DLL (число входов и выходов)
            //-------------------------------------------------------------------------
------
            int inputs;
            int outputs;
            nn.GetInputOutputInfo(inputs,outputs);

            //-------------------------------------------------------------------------
------
            // Step 3:  Загрузите начальные веса сети. (Необязательный)
            //-------------------------------------------------------------------------
------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ: WEIGHTS_PATH_NAME определен в Globals.h
            // отображайте то из файла весов, который сохранялся, когда DLL
генерируется.
            //-------------------------------------------------------------------------
------
            // Примечание: Этот шаг не необходим. Если этот шаг не выполняется,
            // сеть просто начнется со случайных начальных весов.
            //-------------------------------------------------------------------------
------
            nn.LoadWeights(WEIGHTS_PATH_NAME);

            //-------------------------------------------------------------------------
------
            // Снова устанавливает сеть (смешивает веса сети, и т.п.), если потребитель
имеет так
```

```
            // указанный, проверяя коробку чека "Снова Установленная Сеть
Перед Обучением"
            //-----------------------------------------------------------------
------------
            CButton *pResetNetworkCheckBox = (CButton *) GetDlgItem
(IDC_RESET_NETWORK);
            if (pResetNetworkCheckBox->GetCheck() == 1)
                  nn.ResetNetwork();

            //-----------------------------------------------------------------
------------
            // Step 4: Определите начальные данные.
            //-----------------------------------------------------------------
------------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ: Следующее загружает учебные начальные
данные из вашей
            //начальной буквы конфигурация сети в массив inputData.
            // Отметьте, что число входов в начальных данных должно
соответствовать числу входов
            // ожидаемый генерируемым DLL.  Также, может быть какое-нибудь
число образцов в
            // начальные данные, пока число образцов в начальных данных
соответствует числу
            // образцов в желаемых данных. В данном случае, в
NUMBER_OF_EXEMPLARS определяется
            // Globals.h.
            //-----------------------------------------------------------------
------------
            double *inputData = FillArrayFromFile(INPUT_FILE_PATH_NAME,
inputs, NUMBER_OF_EXEMPLARS);

            //-----------------------------------------------------------------
------------
            // Step 5: Определите желаемые данные.
            //-----------------------------------------------------------------
------------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ: Следующее загружает учебные желаемые
данные из вашей начальной буквы
            // конфигурация сети в массив desiredData.
            // Отметьте, что число выходов в желаемых данных должно
соответствовать числу выходов
            // ожидаемый генерируемым DLL.  Также, может быть какое-нибудь
число образцов в
            // желаемые данные, пока число образцов в желаемых данных
соответствует числу
            // образцов в начальных данных. В данном случае, в
NUMBER_OF_EXEMPLARS определяется
            // Globals.h.
            //-----------------------------------------------------------------
------------
            double *desiredData = FillArrayFromFile(DESIRED_FILE_PATH_NAME,
outputs, NUMBER_OF_EXEMPLARS);

            //-----------------------------------------------------------------
------------
            // Step 6: Разрешите автоматическое сохранение лучших весов.
(Необязательный)
            //-----------------------------------------------------------------
------------
```

```
            // Примечание: Этот шаг не необходим.  Значение по умолчанию
этого свойства есть
            // всегда Ложь.  Установка этого к Истине заставит лучшие веса
сети быть
            // сохраняемый в течение обучения.  Если это свойство установлено
к Истине, не забудьте вызвать
            // SetBestWeightsPathName() чтобы установить действительный путь
для лучших весов (Посмотрите Шаг 7).
            //-----------------------------------------------------------------
------------
            nn.SetSaveBestWeightsEnabled(true);

            //-----------------------------------------------------------------
------------
            // Step 7: Установите путь для сохранения лучших весов.
(Необязательный)
            //-----------------------------------------------------------------
------------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ : BEST_WEIGHTS_PATH_NAME определен в
Globals.h.
            //-----------------------------------------------------------------
------------
            // Примечание: Указанный путь будет использоваться для сохранения
лучших весов в течение
            // обучение.  Этот SetBestWeightsPathName() функция должна быть
установлена
            // действительный путь, если SetSaveBestWeightsEnabled()
установлен к истине.  Другой,
            // установка этого свойства необязательна.
            //-----------------------------------------------------------------
------------
            nn.SetBestWeightsPathName(BEST_WEIGHTS_PATH_NAME);

            //-----------------------------------------------------------------
------------
            // Step 8: Обучите сеть.
            //-----------------------------------------------------------------
------------
            // ПРИМЕЧАНИЯ ВЫПОЛНЕНИЯ : NUMBER_OF_EPOCHS и NUMBER_OF_EXEMPLARS
определен
            // в Globals.h и определен от начальной конфигурации сети.
            //-----------------------------------------------------------------
------------
            // Примечание: Две версии Train() функция являются обеспеченными.
Эта версия делает
            // не включайте перекрестных параметров утверждения.  Если
перекрестное утверждение желается,
            // используйте другую версию Train() и вызвать
SetCrossValidationEnabled(истина).
            //-----------------------------------------------------------------
------------
            //int r;
            //for(r=0; r<100; r++)
            //{
            nn.Train(NUMBER_OF_EPOCHS, NUMBER_OF_EXEMPLARS, inputData,
desiredData);
            //}
```

```
            //-----------------------------------------------------------------
------------
            // Step 9: Сообщите лучшую стоимость. (Необязательный)
            //-----------------------------------------------------------------
------------
            // Это основанное на диалоге приложение сообщает лучшую стоимость
потребителю.
            //-----------------------------------------------------------------
------------
            // Примечание: Этот шаг не необходим, это является обеспеченным
только, чтобы показать, какой лучший
            // стоимость может быть получена.
            //-----------------------------------------------------------------
------------
            double bestCost = nn.GetBestCost();
            m_TrainNetwork_Report.Format("Best Cost = %g",bestCost);

            //-----------------------------------------------------------------
------------
            // Perform some memory clean-up
            //-----------------------------------------------------------------
------------
            free (inputData);
            free (desiredData);
      }

      //-----------------------------------------------------------------------
-------
      // If initialization failed, report any errors here.
      //-----------------------------------------------------------------------
-------
      else
      {
            //-----------------------------------------------------------------
------------
            // If the DLL was loaded, this is probably a recall-only network.
            //-----------------------------------------------------------------
------------
            if (nn.IsLoaded())
            {
                  m_TrainNetwork_Report = "DLL recall only.";
            }
            //-----------------------------------------------------------------
------------
            // Else, the DLL was not loaded, the path is probably incorrect.
            //-----------------------------------------------------------------
------------
            else
            {
                  m_TrainNetwork_Report = "DLL load failed.";
            }
      }

      //-----------------------------------------------------------------------
-------
      // Update the dialog.
      //-----------------------------------------------------------------------
-------
      UpdateData(FALSE);
```

```
}


void CVCPPShellDlg::OnEnChangeGetNetworkOutputReport()
{
	// TODO:  If this is a RICHEDIT control, the control will not
	// send this notification unless you override the
CDialog::OnInitDialog()
	// function and call CRichEditCtrl().SetEventMask()
	// with the ENM_CHANGE flag ORed into the mask.

	// TODO:  Add your control notification handler code here
}

void CVCPPShellDlg::OnEnChangeTrainNetworkReport()
{
	// TODO:  If this is a RICHEDIT control, the control will not
	// send this notification unless you override the
CDialog::OnInitDialog()
	// function and call CRichEditCtrl().SetEventMask()
	// with the ENM_CHANGE flag ORed into the mask.

	// TODO:  Add your control notification handler code here
}



void CVCPPShellDlg::OnBnClickedResetNetwork()
{
	// TODO: Add your control notification handler code here
}


void CVCPPShellDlg::OnBnClickedResetNetwork2()
{

}


void CVCPPShellDlg::OnButton3()
{


	//////////////////////////////////////////////////////////////////////
/////

}

// Get the name of the Excel-ODBC driver
CString CVCPPShellDlg::GetExcelDriver()
{
	char szBuf[2001];
	WORD cbBufMax = 2000;
	WORD cbBufOut;
	char *pszBuf = szBuf;
	CString sDriver;

	// Get the names of the installed drivers ("odbcinst.h" has to be
included )
```

```
   if(!SQLGetInstalledDrivers(szBuf,cbBufMax,& cbBufOut))
          return "";

      // Search for the driver...
      do
      {
            if( strstr( pszBuf, "Excel" ) != 0 )
            {
                  // Found !
                  sDriver = CString( pszBuf );
                  break;
            }
            pszBuf = strchr( pszBuf, '\0' ) + 1;
      }
      while( pszBuf[1] != '\0' );

      return sDriver;
}


void CVCPPShellDlg::OnRadio1()
{
      // TODO: Add your control notification handler code here

}
```