МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Центральноукраїнський національний технічний університет

Кафедра кібербезпеки та програмного забезпечення

На правах рукопису

Нужний Дмитро Олегович

Програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів

Спеціальність: 123 «Комп'ютерна інженерія»

Освітній ступінь: бакалавр

Науковий керівник:

Смірнов Олексій Анатолійович _____

(підпис) (дата)

доктор технічних наук, професор

ДОПУЩЕНО ДО ЗАХИСТУ

Завідувач кафедри

_____ О.А. Смірнов

(підпис) ПІБ

«____»_____2021 р.

Кропивницький – 2021

Міністерство освіти і науки України
Центральноукраїнський національний технічний університет

Центр *Заочної та дистанційної освіти*
Кафедра *Кібербезпеки та програмного забезпечення*
Освітній ступінь *бакалавр*
Спеціальність *123 Комп'ютерна інженерія*

ЗАТВЕРДЖУЮ
Завідувач кафедри
д.т.н., проф.
_____О.А.Смірнов
« 11 » січня 2021 року

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

Нужному Дмитру Олеговичу
(прізвище, ім'я, по батькові)

1. Тема роботи *Програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів*

керівник роботи *Смірнов Олексій Анатолійович, докт. техн. наук, професор*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу № 186-02 від 28.12.2020 року

2. Строк подання студентом роботи до захисту *22.05.2021 р.*

3. Мета та завдання кваліфікаційної бакалаврської роботи*: Метою розробки є програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Призначення та область використання.
2.Перегляд аналогічних існуючих систем.
3. Опис і обґрунтування проектних рішень.
4. Етапи програмування системи.
5. Впровадження системи в промислову експлуатацію.
6. Висновки

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)

Структурна схема системи	*1 аркуш*
Функціональна схема системи	*1 аркуш*
Діаграма процесів	*1 аркуш*
Блок-схема алгоритму роботи додатку	*2 аркуша*

6. Дата видачі завдання « 11 » січня 2021 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної бакалаврської роботи	Строк виконання етапів кваліфікаційної бакалаврської роботи	Примітка
1.	Аналіз існуючих систем	10.03.2021 р.	
2.	Постановка задачі, оформлення ТЗ	15.03.2021 р.	
3.	Розробка моделі компонента	20.03.2021 р.	
4.	Розробка структур даних	25.03.2021 р.	
5.	Розробка алгоритмів зв’язку та відображення	30.03.2021 р.	
6.	Програмування алгоритмів	10.04.2021 р.	
7.	Оформлення ПЗ	17.04.2021 р.	
8.	Попередній захист роботи	14.05.2021 р.	

Студент _____ _____
(підпис) (прізвище та ініціали)

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

АНОТАЦІЯ

Нужний Д.О. Програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів. 123 Комп'ютерна інженерія. Центральноукраїнський національний технічний університет. Кропивницький. 2021.

В даній кваліфікаційній бакалаврській розроблено програмне забезпечення, яке призначено для системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

Метою розробки є програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

Результат роботи – програмна реалізація системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

В процесі роботи над програмною моделлю виконано аналіз існуючих апаратних та програмних засобів. В повній мірі описані всі компоненти розробленого програмного забезпечення.

Розроблено зручний інтерфейс користувача. Наведені інструкції по роботі з програмними засобами.

Програма може використовуватися на ПЕОМ архітектури IBM PC з ОС Windows XP/Vista/7/8/10.

Програму розроблено в середовищі RAD Studio Delphi 10.4.

Ключові слова: комп'ютерна інженерія, пошук інформації, нейронні мереж

ABSTRACT

Nuzhnyi D.O. Software for information retrieval system using neural network technology with transformer architecture. 123 Computer Engineering. Central Ukrainian National Technical University. Kropyvnytskyi. 2021

In this bachelor's qualification the software which is intended for system of search of the information by means of technology of neural networks with architecture of transformers is developed.

The purpose of the development is the software of information retrieval system using neural network technology with transformer architecture.

The result is a software implementation of information retrieval system using neural network technology with transformer architecture.

In the process of working on the software model, an analysis of existing hardware and software was performed. All components of the developed software are fully described.

Developed user-friendly interface. Instructions for working with software are given.

The program can be used on an IBM PC with Windows XP / Vista / 7/8/10.

The program is developed in the environment of RAD Studio Delphi 10.4.

Keywords: computer engineering, information retrieval, neural networks

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ І ТЕРМІНІВ

ААУ	–	автономне адаптивне управління
БД	–	блок датчиків
БЗ	–	база знань
БОС	–	блок оцінки стану
БПР	–	блок прийняття рішень
ГІП	–	графічний інтерфейс користувача (GUI)
ВО	–	виконавчий орган
ЕОМ	–	електрона обчислювальна машина
КА	–	кінцевий автомат
НРС	–	недетермінований автомат Рабина-Скотта
НМ	–	нейронна мережа
МНРС	–	модифікований недетермінований автомат Рабина-Скотта
НАНУ	–	Національна академія наук України
ОУ	–	об'єкт управління
ПЧО	–	просторово-часовий образ
ВВ	–	випадкова величина
ПЗ	–	програмне забезпечення
СПДНМ	–	система побудови й дослідження нейронних мереж
СРНМ	–	Система розробки нейронних мереж
УС	–	управляюча система
ФР	–	функція розподілу
ФРО	–	апарат формування й розпізнавання образів
Z^+	–	множина ненегативних цілих чисел
$G(V, N)$	–	граф із множиною вершин V і множиною ребер N
$i \to j$	–	ребро, спрямоване з вершини i у вершину j
$X \leftrightarrow Y$	–	взаємооднозначне відображення множини X на множину Y

$\pi(X)$	–	множина кінцевих підмножин множини X
$R[a,b]$	–	множина речовинних чисел на $[a,b]$, $R \equiv R[-\infty,+\infty]$
B^N	–	простір двійкових векторів розмірності N
Λ	–	порожнє слово із множини вхідних слів КА
0	–	*неправда* у вираженні тризначної логіки
1	–	*істина* у вираженні тризначної логіки
$\Diamond$	–	*невизначеність* у вираженні тризначної логіки
$\vec{X} \subset \vec{Y}$	–	$\vec{X}$ є підвектор (сукупність обраних компонентів) вектора $\vec{Y}$
$X \supset Y$	–	клас Y є нащадком класу X

ВСТУП

Актуальність теми. У даній роботі розробляється пошуковий сервіс на базі передової технології аналізу текстів YATI, у якій задіяні нейронні мережі нового покоління – трансформери. Це загальна назва архітектури, яка лежить в основі сучасних підходів до аналізу тексту. YATI розшифровується як Yet Another Transformer with Improvements – «Ще один трансформер з поліпшеннями».

Трансформери являють собою надвеликі й надскладні нейронні мережі. Вони дуже добре справляються із самими різними завданнями в сфері обробки природньої мови, від машинного перекладу до генерації текстів, але вимагають багато обчислювальних ресурсів. Зрівняєте самі: нейромережа, яка використовувалася раніше, навчалася на одному графічному прискорювачі, і процес навчання займав годину. Якщо брати той же прискорювач і почати навчати на ньому більшу нейромережу-трансформер, на це піде десять років. Тому впровадження трансформерів – непросте інженерне завдання. Щоб розгорнути YATI, треба було об'єднати багато прискорювачів у кластери, зв'язати їх у мережу й розробити для серверів, що вийшли, потужну систему охолодження.

У пошуковій системі YATI зіставляє зміст запитів і веб-документів. YATI показує набагато кращі результати чому інші підходи за рахунок переваг, які є в трансформерів. Вони вміють працювати не тільки з короткими, такими як запити або заголовки статей, але й з довгими текстами. У них є «механізм уваги», який дозволяє виділяти в тексті самі значимі фрагменти. Нарешті, трансформери звертають увагу на порядок слів і враховують контекст – те, як слова впливають один на одного. У багатьох випадках порядок слів визначає зміст усієї фрази.

Мета й завдання дослідження. Метою роботи є програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

Для досягнення поставленої мети визначена програма дослідження, що складається з наступних завдань:

– Огляд існуючих систем пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

– Дослідження системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

– Програмна реалізація системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

Практична цінність отриманих результатів полягає в тому, що розроблені алгоритми дозволяють успішно вирішувати задачі пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

1 ПРИЗНАЧЕННЯ ТА ОБЛАСТЬ ВИКОРИСТАННЯ

1.1 Призначення системи

Робота призначена для пошукової системи в Інтернеті. Пошукова система (пошукач) – це сайт, який здійснює пошук відповіді на запит користувача по всім відомих даній системі сайтам.

Усі пошукові системи використовують власні алгоритми? побудови списку сайтів, що містять на думку пошукача відповідь на запит користувача. Крім алгоритмів пошукова система використовує в роботі роботів?, які індексують сайти, зображення, перевіряють доступність сайтів та ін.

Пошук може здійснюватися не тільки по текстовому запиту, що вводиться в рядок пошуку, але, наприклад і по картинці або голосовому повідомленні.

Більшість пошукових систем ураховують регіональність сайту, користувача і його запиту, видаючи відвідувачеві найбільш коректний на думку пошукача відповідь у вигляді списку сайтів.

Види пошукових систем

Пошукові системи діляться на наступні види:

– Національні пошукові системи. Розробляються споконвічно для пошуку сайтів усередині конкретної країни, тобто для внутрішнього ринку. Більшість із них поступово вийшли за рамки своєї держави, але при цьому не перейшли в розряд транснаціональних. Приклад національних пошукових систем: Cade (br), Alcanseek (cn), Alexa (us), Anzwers (au), ...

– Транснаціональні пошукові системи. Здійснюють пошук відповіді на запит користувача по сайтах усіх країн, незалежно від їхньої доменної зони й країни знаходження. Приклад транснаціональних пошукачів: Google, Baidu, Yahoo!, Bing.

1.2 Область застосування

У даному розділі перелічимо пошукові системи по країнах світу

Австралія й Нова Зеландія:

– Access New Zealand.
– Altavista.
– Anzwers.
– AOL.
– The Aussie Index.
– Google.
– Ninemsn.
– Nzpages.
– Sensis.
– Webwombat.
– Yahoo.

Бельгія:

– Webwatch.

Бразилія:

– Cade.
– Google Brasil.
– MSN Brasil.
– Yahoo Brasil.

Канада:

– Alcanseek.
– Altavista Canada.
– British Columbia Community Pages.
– Canada.com.
– Maple Square.
– National Library of Canada.

– Toutmontreal.com.
– Search Canada.
– Ontario Online.
– Ontario Web Guide.
– Yahoo Canada.

Китай:

– Baidu.
– Google China.
– MSN China.
– Hksrch.
– Yahoo! Chinese.
– Yahoo! Hong Kong.
– Yisou.com.

Данія:

– Jubii.

Франція:

– Cocorico!
– France Pratique.
– Google France.
– Lycos France.
– MSN France.
– Nomade.
– Sharelook France.
– Yahoo France.

Германія:

– Adguide.
– Aladin.
– Altavista Germany.
– bellnet.com.

– Crawler.De.
– Dino.
– Fireball.
– Focus Netguide.
– Google Deutschland.
– Kostenlos.
– LEO – Link Everything Online.
– Lycos Germany.
– MSN Deutschland.
– Sharelook Deutschland.
– Suchmaschine.com.
– Web.de.
– Yahoo Germany.
– Yellowmap.

Ізраїль:

– Maven.

Італія:

– Abacho Italia
– Google Italia.
– Il Trovatore.
– Multisoft Italiano.
– Virgilio.
– Yahoo! Italia.

Ямайка:

– Caribseek Jamaica.
– Netsearch Jamaica.
– Top 5 Jamaica.

Японія:

– Google Japan.

– Goo.
– Infoseek Japan.
– Lycos Japan.

Кенія:

– Kenyaweb Search.

Мексика:

– Explora Mexico.
– Mexico Web Guide.
– Radar.

Нідерланди:

– Lycos Nederland.
– Zoek.

Норвегія:

– Fast.

Португалія:

– AEIOU.
– Sapo.

ПАР:

– Aardvark.
– Ananzi South Africa.
– Funnel.
– South Africa Online.
– South Africa Web-Chart.

Іспанія:

– Google España.
– Hispavista.
– MSN España.
– Ozu.
– Terra.

– Ya.com.
– Yahoo España.
– Wannadoo.

Швеція:

– Lycos Sverige.

Швейцарія:

– Lycos Switzerland.
– Swiss Search.
– The Blue Window.

Англія:

– Britindex.
– Google UK.
– Interview.
– Lifestyle.UK.
– Searchuk.
– UK Plus.
– Yahoo UK & Ireland.
– Yell.

США:

– A9.
– Alexa.
– Alltheweb.
– Altavista.
– AOL Search.
– Ask Jeeves.
– Blowsearch.
– Business.com.
– Clusty.
– DMOZ.

– Excite.
– Gigablast.
– Google.
– Hotbot.
– iwon.
– Looksmart.
– Lycos.
– MSN.
– Netscape.
– Northern Light.
– Overture / Yahoo! Search Marketing.
– Teoma.
– What-U-Seek.
– Yahoo.

Таким чином, виходячи з вищеперерахованого, програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів, є актуальною задачею, яка потребує вирішення у даній кваліфікаційній бакалаврській роботі.

2 ПЕРЕГЛЯД АНАЛОГІЧНИХ ІСНУЮЧИХ СИСТЕМ

2.1 Огляд існуючих систем, технологій, архітектур, програмних рішень за профілем теми кваліфікаційної бакалаврської роботи

Google

Згідно даним Statcounter за травень 2021 року, Google займає 92.04% ринку пошукових систем. Ці дані не включають соціальні мережі (Facebook, YouTube, Twitter і т.д.).

Google намагається дати найкраща відповідь на запит користувача. Саме із цією метою й відбувається регулярна еволюція його алгоритмів. Але цей процес завів Google до цікавого результату.

Часто фактори пов'язані з авторитетом домену, трастом сайту, кількістю зворотних посилань мають більше значення, ніж цінність інформації для користувача. Як результат, сайти, які краще відповідають на запит користувача, але не мають більшого «авторитету» в очах пошукової системи, залишаються поза увагою користувача. Дивно, але одна із самих інноваційних компаній у світі робить свої алгоритми консервативніше. Просунуті користувачі (як читачі, так і автори контенту) зауважують це і їх усе більше дратує монополія Google серед пошукових систем.

Крім того, багато знають, що Google детально відслідковує кожний крок користувача. Крім розуміння, як поліпшувати свої продукти, це також дає можливість Google створювати кращі умови для таргетингу в рекламній системі Google Ads. Хтось із нас не обертає на це уваги. Але усе більше росте кількість користувачів, які прагнуть використовувати пошукові системи без рекламних оголошень, або ж шукати інформацію з інших алгоритмів. На щастя, існує безліч альтернатив для Google. Кожна з них має певні переваги й особливі сфери застосування.

Важливо уточнити, що дана стаття створена не для того, щоб занизити значення гугла в очах активних користувачів усієї мережі. Її ціль, лише показати велика кількість інструментів і варіантів пошуку інформації, які зараз існують. Google не був першопрохідником на ринку пошукових систем, але він (а саме Сергій Брін і Ларрі Пейдж) створив самий просунутий для свого часу алгоритм ранжирування сайтів. І протягом усієї історії розвитку ринку пошукових систем, він визначав і визначає правила гри на цьому ринку.

Bing

Bing є самою популярною пошуковою системою після Google (Yahoo був викуплений компанією Microsoft в 2009 році). Це єдина пошукова система Microsoft, яка була розроблена спеціально для систем Windows.

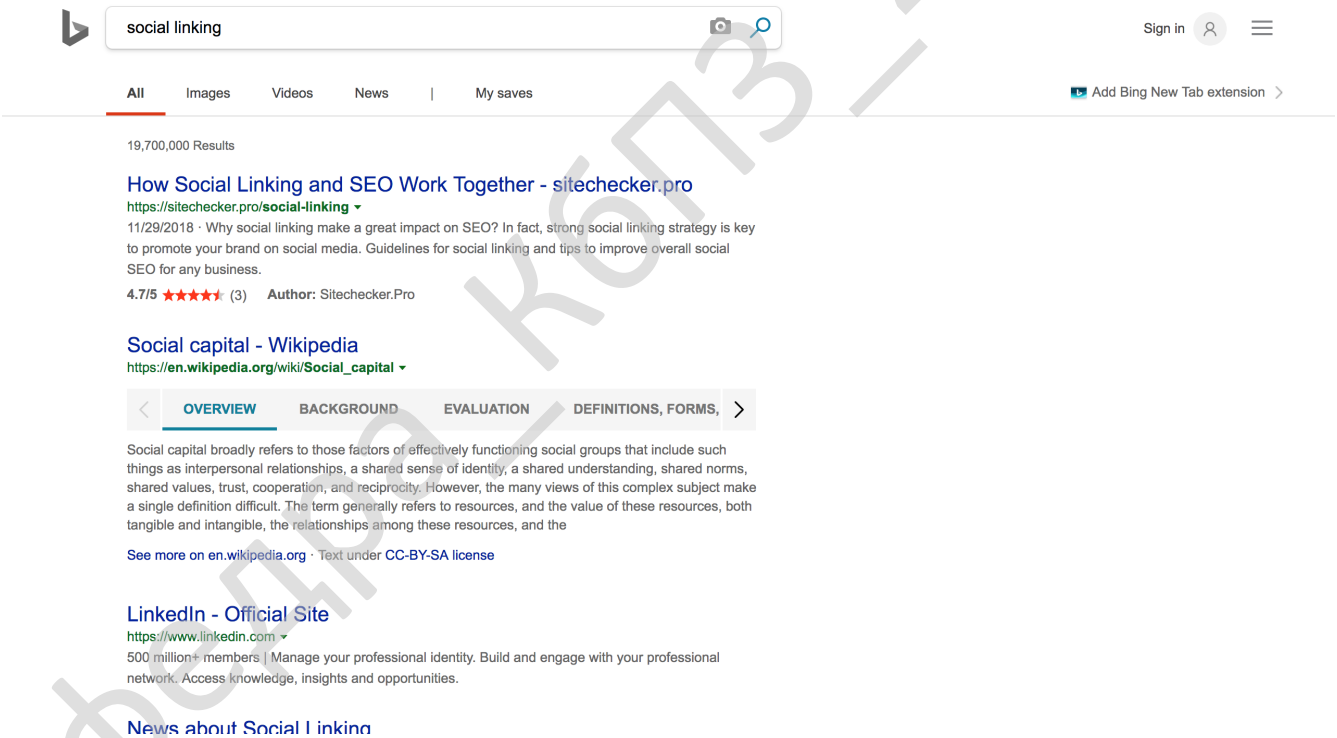

Рисунок 2.1 – Інтерфейс користувача Bing

В Bing є відмінні можливості пошуку відео, які навіть краще, чим в Google. Тут більше параметрів автозаповнення, при введенні запитів користувача.

Він відслідковує більше взаємозв'язків між окремими веб-сайтами, і завдяки цьому пошук в інтернеті схожих варіантів спрощується.

Duckduckgo

Duckduckgo – ще один популярний варіант пошуку, який передвстановлений у деяких популярних браузерах (наприклад Firefox). Це один із кращих варіантів для тих, хто не прагне, щоб їх дані відслідковувалися. Звичайно Duckduckgo протиставляється Google, який схожий на «Великого брата» і стежить за кожним кроком користувача.

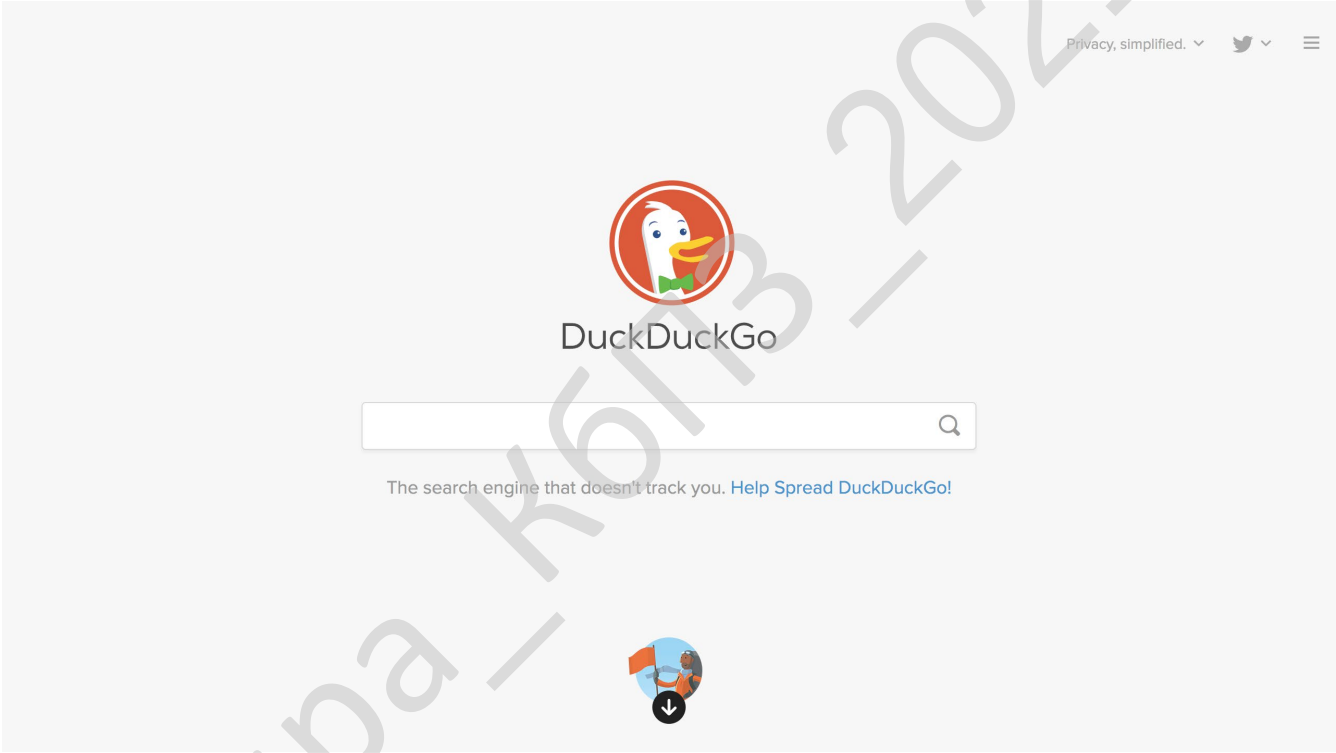

Рисунок 2.2 – Інтерфейс користувача Duckduckgo

Boardreader

Boardreader сподобається тим, хто цікавиться незвичайними пошуковими системами. Він розроблений як проста дошка оголошень і шукає результати винятково на форумах по усьому світу, де реальні люди діляться своїм досвідом по зазначеній темі.

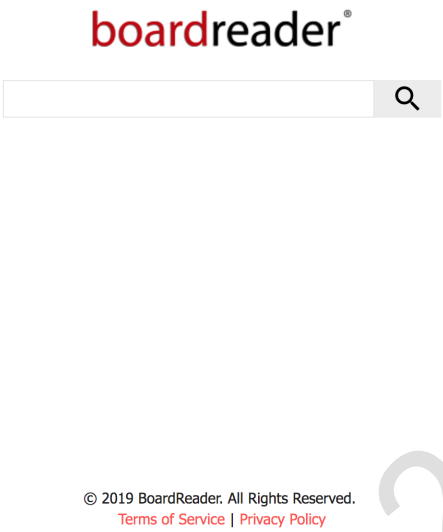

Рисунок 2.3 – Інтерфейс користувача Boardreader

Dogpile

Dogpile – справжній комбайн для збору даних, оскільки він сканує результати пошуку трьох популярні пошукові систем (Google, Yandex і Yahoo), і вибирає кращі результати з кожної. Простота інтерфейсу й відсутність рекламних оголошень дозволяє краще зосередитися на пошуку.

Creative Commons Search

Creative Commons Search або в скороченому варіанті CC Search – унікальна анонімна пошукова система. Вона дає можливість одержувати авторські матеріали із правами для повторного використання в особистих цілях. Якщо ви плануєте створити персональну веб-сторінку, CC Search стане відмінним місцем для збору дизайнерських матеріалів.

Giphy

Giphy – ідеальна пошукова система для тих, хто захоплений анімаційними картинками у форматі GIF. Вона була спеціально розроблена для пошуку

мініатюрних відеороликів. Тут можна одержати багато позитивних емоцій, при пошуку смішних кошенят або веселих ситуацій з нескінченним повторенням.

Quora

Quora більше схожа на інформаційний портал, чому на пошукову систему. Тут можна спілкуватися з людьми на різноманітні теми й одержувати відповіді на важливі питання від користувачів по усьому світу. Більше того, на сайті є спеціальна категорія тематичних питань, які задавалися раніше. Україномовним аналогом такого сервісу є Thequestion. У них різний дизайн і структура, але суть одна – користувач прагне одержати кращу відповідь на своє питання від реальних людей.

Vimeo

YouTube – це підрозділ Google, наповнене безліччю рекламних оголошень. Vimeo – популярна альтернатива, яка дуже зручна для використання й побудована на простій системі обміну відео. Найкраще в цьому сервісі – повна відсутність реклами й велика вибір HD-відео.

WolframAlpha

WolframAlpha найімовірніше сподобається комп'ютерним гикам. дизайн, що виділяється, велика кількість додаткових функцій, свої нестандартні алгоритми пошуку.

StartPage

StartPage – ще один анонімний сервіс для людей, які божевільні на своїй конфіденційності й негативно сприймають політикові передачі секретних даних Google для комерційного використання. Він не тільки дозволить провести абсолютно безпечний пошук, але також сховає ваші дані, такі як IP і MAC-адреси. Це дозволяє заходити на будь-які веб-сайти через спеціальний проксі-сервер і не залишати слідів присутності користувача на сайтах.

Ask.com

Ask.com – сервіс, який поєднує всі популярні пошукові системи й генерує спеціальні тематичні сторінки, де користувачі можуть додавати в закладки

найцікавіші результати пошуку. Крім того, тут ви можете попросити людей про додаткову раду.

Slideshare

Slideshare стане відмінним джерелом для пошуку корисних матеріалів: презентацій, інфографік, документів. У цього сайту є необмежена база презентацій, яка доступна для всіх зареєстрованих користувачів (і вона постійно обновляється).

2.2 Обґрунтування вибору засобів для побудови системи та мови програмування

Embarcadero Delphi, раніше Borland Delphi і Codegear Delphi, – інтегроване середовище розробки ПЗ для Microsoft Windows, MAC OS, iOS і Android мовою Delphi (що раніше носила назву Object Pascal), створена спочатку фірмою Borland і на даний момент приналежна й розроблювальна Embarcadero Technologies. Embarcadero Delphi є частиною пакета Embarcadero RAD Studio і поставляється в чотирьох редакціях: Community (поширюється безкоштовно й має обмежену ліцензію на використання в комерційних цілях), Professional, Enterprise і Architect.

Delphi 10.4 Sydney

Випущено 26 травня 2020 року. RAD Studio Delphi 10.4 забезпечує значно поліпшену високопродуктивну нативну підтримку Windows, кращу продуктивність розробки, миттєві підказки code completion, прискорення виконання коду із синтаксисом керованих записів, поліпшення виконання паралельних завдань на сучасних багатоядерних CPU, а також містить більш 1000 виправлень багів, поліпшення продуктивності середовища й бібліотек і багато чого крім того.

Основні можливості Delphi 10.4.1:

− Істотні розширення для Windows: поліпшення для застосунків на моніторах 4K High DPI, інтеграція з новим WebView2 на базі Chromium,

використання розширених title bars, таких же, як в Office, Explorer, Google Chrome.

− Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

− Істотне поліпшення Delphi Code Insight (без можливого блокування IDE – в окремому процесі), що допоможе при роботі з великими проектами.

− Тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання.

− Розширена підтримка бібліотек C++: ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode.

− Відладник Win 64 (на LLDB) і збирач для C++.

− Поліпшення для C++: Включена велика кількість поліпшень STL з Dinkumware.

− Підтримка Metal Driver GPU для macOS і iOS.

− Вбудований Fmxlinux.

− Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API. Реалізація компонента Media Player для macOS тепер використовує Avfoundation. Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

− Численні поліпшення швидкості й стабільності роботи нашої бібліотеки The Parallel Programming Library (PPL).

− Додані оновлені драйвери для FireBird, PostgreSQL і SQLite.

− Клієнтські бібліотеки HTTP і REST Client розширені застосунковими можливостями роботи з HTTPS. Також були розширені можливості підтримки Amazon AWS services

− У технологію Visual LiveBindings внесена безліч поліпшень, у тому числі швидкодії, що стосуться, застосунків на VCL і FireMonkey

RAD Studio 10.4 Короткий огляд:

– Істотні розширення для Windows. Створення застосунків, що чудово виглядають, із чіткими елементами інтерфейсу на 4k моніторах High DPI за допомогою нової гнучкої підтримки стилів елементів керування на екрані. Інтеграція із сучасними, безпечними web-технологіями від Microsoft – новим WebView2 на базі Chromium. Використання сучасних розширених title bars, таких же, як в Office, Explorer, Google Chrome, у своїх проектах. Істотні поліпшення надійності налагодження в новому відладнику для C++ Windows 64-bit.

– Зросла продуктивність розробки. Ріст продуктивності за рахунок миттєвої реакції підказок code completion у середовищі IDE. Краща сумісність із уже наявною кодовою базою, і спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю. Швидке зв'язування даних і візуальних елементів за допомогою розширеної технології Visual LiveBindings з підвищеною швидкодією. Просте використання розповсюджених бібліотек C++, наприклад, ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode. Оновлена підтримка Amazon AWS cloud.

– Поліпшення швидкодії і якості. Більш 1000 поліпшень швидкодії і якості. Краща ефективність коду за допомогою нового синтаксису custom managed records. Більш швидке виконання паралельних завдань на сучасних багатоядерних CPU. Переконаєтеся в прискоренні відображення на екрані з підтримкою Metal API на macOS і iOS. Краща сумісність із уже наявною кодовою базою й спрощення програмування за рахунок уніфікованої архітектури керування пам'яттю.

Істотне поліпшення Delphi Code Insight

Як найбільше й головне поліпшення інструментів програмування Delphi за багато років, в 10.4 Delphi Code Insight реалізований через Language Server Protocol (LSP). LSP – це технологія генерації результатів для code completion, навігації й інших сервісів в окремому процесі. Це значить, що code completion і Code Insight одержать більш точні результати без блокування IDE. 10.4

забезпечує набагато більш високу продуктивність розроблювачів, які працюють із більшими проектами, що містять мільйони рядків коду.

Delphi Custom Managed Records

Ключове розширення мови Delphi: тип даних Delphi «record» тепер підтримуть довільні ініціалізацію, фіналізацію й операції копіювання. Управляйте тем, як ці структури створюються, копіюються й звільняються з допомогу вашого коду, який буде виконуватися у відповідний момент.

Це розширює потужність конструкцій records в Delphi, які використовуться щоб одержати більшу ефективність у порівнянні із класами.

Єдине керування пам'яттю

Керування пам'яттю в Delphi тепер стандартизоване на всіх підтримуваних платформах – мобільних, настільних і серверних – використовучи класичну реалізацію керування пам'яттю об'єктів.

У порівнянні з Automatic Reference Counting (ARC), це дає кращу сумісність із існучим кодом і спрощує написання компонентів, бібліотек і застосунків.

ARC модель керування пам'яттю model залишилася для керування рядками й посиланнями на тип інтерфейсу на всіх платформах. Для C++ це означає, що при створенні й звільненні Delphi-style класів в C++ використовується звичайне керування пам'яттю, як у будь-якого heap-allocated класу C++, що значно знижує складність коду.

Розширена підтримка бібліотек C++

В 10.4 ми портували багато популярних бібліотек C++ у C++Builder.

Забезпечивши оптимізовану підтримку бібліотек ZeroMQ, SDL2, SOCI, libSIMDpp і Nematode, поряд із уже підтримуваними Boost і Eigen, які можуть бути додані за допомогою менеджера пакетів Getit.

Win 64-відладник і збирач для C++

В 10.4 з'явився новий відладник C++ для Windows 64-bit. Відладник заснований на LLDB і показує значне збільшення стабільності при налагодженні

64-bit застосунків поряд з новими відладочними можливостями, такими як перегляд і інспекція типів начебто рядків C++ і Delphi, а також колекцій STL, включаючи std::vector, std::map і інших. Крім того, згенерована для застосунку відладочна інформація має інший внутрішній формат, сприяючи більш стабільному й багатому на можливості процесу налагодження, більш докладним перегляду й інспекції в debug-time.

Підвищення якості й швидкодії інструментів

- Велика кількість поліпшень STL від Dinkumware.
- Поліпшені деякі найважливіші методи й області RTL, на базі поліпшень сумісності з популярними бібліотеками C++.
- Поліпшена підтримка Cmake.
- Велика кількість виправлень для підвищення стабільності і якості.
- Відновлення Windows API – Обновлено й додали безліч декларацій API щоб добитися ще більшої інтеграції із платформою Windows.
- Загальні вдосконалення в бібліотеці доступу до БД FireDAC, включаючи оновлені драйвера для FireBird, PostgreSQL і SQLite. Вибір статичного або динамічного підключення SQLite до застосунку.

Змінені стилі VCL для High DPI

В 10.4, архітектура стилізації VCL була суттєво розширена для підтримки High DPI і 4K моніторів. Тепер усі елементи UI на формі VCL автоматично масштабуться під відповідне до монітора дозвіл для показу форми. Був оновлений API стилізації для підтримки стилів high DPI.

Кожний графічний елемент UI може бути обраний з наборів різних масштабів і масштабований до потрібного DPI, що дає чітке зображення елементів UI на всіх моніторах.

Нові High DPI стилі й стилізація окремих VCL компонент

Обновлено велике число вбудованих і преміальних VCL стилів для підтримки нового режиму стилізації High-dpi. Це дозволяє вам створювати застосунку з відмінним дизайном для всіх моніторів.

Розроблювачі VCL застосунків тепер можуть використовувати трохи VCL стилів на різних формах в одному застосунку або в різних компонентів на одній формі. Це також включає стилізацію компонентів загальною темою для платформи. Крім застосункової гнучкості використання стилів, це дозволяє використовувати нестилізуємі компоненти із зовнішніх бібліотек в VCL застосунках, що використовуть стиль.

Поліпшена кроссплатформеність

– Додана підтримка Metal Driver GPU для macOS і iOS.

– Крім підтримки останнього iOS SDK, в RAD Studio 10.4 розроблювачі можуть задовольнити нові вимоги Apple до набору стартових екранів.

– Реалізований заново стилізуємий FMX компонент TMemo на платформі Windows значно поліпшений і тепер має відмінну підтримку IME.

– Користувачам редакцій Enterprise або Architect доступна повна інтеграція Fmxlinux з IDE для створення клієнтських застосунків Linux з GUI.

– Компонент Twebbrowser для iOS тепер реалізований на Wkwebview API.

– Реалізація компонента Media Player для macOS тепер використовує Avfoundation.

Оновлений менеджер пакетів Getit

Менеджер пакетів Getit в IDE був значно вдосконалений.

Дати випуску релізів пакетів тепер видні, і можливе сортування списку по цих датах; відбір тільки встановлених пакетів, контенту, доступного тільки при наявності підписки, багато чого іншого.

Універсальний інсталятор для установки Online і Offline

В 10.4 включений новий універсальний інсталятор, який використовує технологію на базі Getit. Цей інсталятор підтримує як online, так і offline (з ISO) варіанти установки.

Тепер обоє варіанта установки дозволяють вам указати початковий набір можливостей RAD Studio для установки, наприклад, свою комбінацію мов програмування й цільових платформ, мов інтерфейсу, і додавати до нього або видаляти непотрібне в будь-який момент.

2.3 Розгорнута постановка завдання

Згідно з технічним завданням на кваліфікаційну бакалаврську роботу, реалізації підлягає програмне забезпечення, яке призначено для системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

В процесі розробки кваліфікаційної бакалаврської роботи необхідно виконати наступний обсяг роботи:

а) провести аналіз існуючих систем-аналогів для виявлення їх позитивних і негативних якостей. Результати аналізу врахувати в подальших розробках;

б) вибрати та обґрунтувати методику побудови системи контролю роботи технологічного обладнання на виробництві в автоматизованому режимі. Розробити функціональну та структурну схеми системи;

в) розробити програмне забезпечення системи, що дозволить реалізувати поставлену технічним завданням задачу. Побудувати блок-схеми алгоритмів програми та підпрограми;

г) організувати інтерфейс користувача з метою формування та виводу на екран ЕОМ повідомлень про некоректні дії користувача та нестандартні ситуації в роботі технологічного обладнання;

д) розробити рекомендації по організаційних та методичних заходах, які забезпечать впровадження системи в промислову експлуатацію та її подальшу успішну експлуатацію;

е) провести розрахунки по визначенню економічної ефективності розробленої системи;

ж) розробити заходи по охороні праці при впровадженні та експлуатації системи, а також розробити заходи з цивільного захисту;

з) сформувати висновки про виконаний обсяг робіт та одержані результати.

3 ОПИС І ОБҐРУНТУВАННЯ ПРОЕКТНИХ РІШЕНЬ

3.1 Опис функціонування системи

Завдання пошуку

Щоб добре ранжувати результати пошуку, треба навчитися оцінювати семантичну (тобто значеннєву) зв'язок між запитом користувача й документом (веб-сторінкою) з інтернету. Інакше кажучи, потрібно побудувати алгоритм, який зможе передбачити, чи містить даний документ відповідь на запит користувача, є чи в ньому релевантна запиту інформація.

З погляду користувача це зовсім природнє завдання. Зміст запиту йому очевидний; далі досить лише прочитати документ, зрозуміти його й або знайти в ньому потрібний зміст (тоді документ релевантний), або немає (не релевантний).

Алгоритм, на відміну від людини, оперує словами із запиту й тексту документа чисто математично, як рядками із символів. Наприклад, алгоритмічно легко порахувати число співпадаючих слів у запиті й документі або довжину самої довгого підрядка із запиту, який присутній й у д окументі. Можна також скористатися накопиченою історією пошуку й дістати із заздалегідь підготовленої таблиці відомості про те, що запит уже задавався в пошук багато раз, а документ одержав по ньому багато кліків (або навпаки, одержав їх дуже мало, або взагалі ще жодного разу не був показаний). Кожний такий розрахунки приносить корисну інформацію про наявність семантичного зв'язку, але сам по собі не опирається на яке-небудь значеннєве розуміння тексту. Наприклад, якщо документ відповідає на запит, те правдоподібно, що в запиту й документа повинні бути загальні слова й підрядки. Алгоритм може використовувати цей статистичний факт, щоб краще оцінити ймовірність значеннєвого зв'язку, а на основі великої кількості таких розрахунків уже можна побудувати досить гарну

модель. При її використанні на практиці в людини буде виникати відчуття, що алгоритм «розуміє» зміст тексту.

По такому принципу й працював пошук до 2016 року. За роки розробки для підвищення якості була придумана безліч дотепних евристичних алгоритмів. Одних тільки способів порахувати загальні слова запиту й документа було запропоновано й впроваджено кілька десятків.

Тут кожному «хіту» t (тобто входженню слова із запиту в документ, від англ. hit, влучення) привласнюється вага з урахуванням частотності слова в корпусі (IDF, Inverse Document Frequency) і відстаней до найближчих входжень інших слів запиту в документ ліворуч і праворуч по тексту (Leftdist і Rightdist). Потім отримані значення підсумуються по всіх знайдених хітах. Кожна така евристика при впровадженні дозволяла одержати невеликий, але статистично значимий приріст якості моделі, тобто ледве краще наблизити той самий значеннєвий зв'язок. Сумарний ефект від простих факторів поступово накопичувався, і впровадження за тривалий період (півроку-рік) уже помітно впливали на ранжирування.

Небагато інший, але теж добре працюючий спосіб принести якість за допомогою простого алгоритму – придумати, які ще тексти можна використовувати в якості «запиту» і «документа» для вже існуючих евристик. Слова запиту можна розширити близькими їм за змістом (сінонимічними) словами або фразами. Або замість вихідного запиту користувача взяти інший, який сформульований інакше, але виражає схожу інформаційну потребу. Наприклад, для запиту [відпочинок на чорному морі] схожими можуть бути:

[чи можна купатися в чорному морі влітку]

[подорож по чорному морю]

[міста й селища на узбережжя чорного моря]

(Це реальні приклади розширень запиту, узяті з пошуку.)

Як знайти схожі слова й запити – тема окремої роботи. Для цього в нашому пошуку теж є різні алгоритми, а розв'язку завдання дуже допомагають

логи запитів, які нам задають користувачі. Зараз важливо відзначити, що якщо схожий запит знайдений, те його можна використовувати у всіх евристичних розрахунках відразу; досить взяти вже готовий алгоритм і замінити в ньому один текст запиту на інший. Таким методом можна одержати відразу багато корисної інформації для ранжирування. Що може бути ледве менш очевидно – аналогічно запиту можна «розширювати» і документ, збираючи для нього «альтернативні» тексти, які називають словом стріми (від англ. stream). Наприклад, стрім для документа може складатися із усіх текстів вхідних посилань або із усіх текстів запитів у пошук, по яких користувачі часто вибирають цей документ на видачі. Точно так само стрім можна використовувати в будь-якому готовому евристичному алгоритмі, замінивши на нього вихідний текст документа.

Говорячи більш сучасною мовою, розширення й стріми – це приклади додаткових контентних ознак, які пошук уміє асоціювати із запитом і документом. Вони нічим не відрізняються від звичайних числових або категоріальних ознак, крім того, що їх уміст – це неструктурований текст. Цікаво, що евристики, для яких ці ознаки споконвічно розроблялися, уже втратили більшу частину своєї актуальності й корисності, але самі розширення й стріми продовжують приносити користь, тільки тепер уже в якості входів нових нейронних мереж.

3.2 Розробка структурної схеми

У результаті багатьох років роботи над якістю пошуку нагромадилися тисячі самих різних факторів. При розв'язку завдання ранжирування всі вони подаються на вхід однієї підсумкової моделі. Для її навчання ми використовуємо відкриту реалізацію алгоритму GBDT (Gradient Boosting Decision Trees) – Catboost.

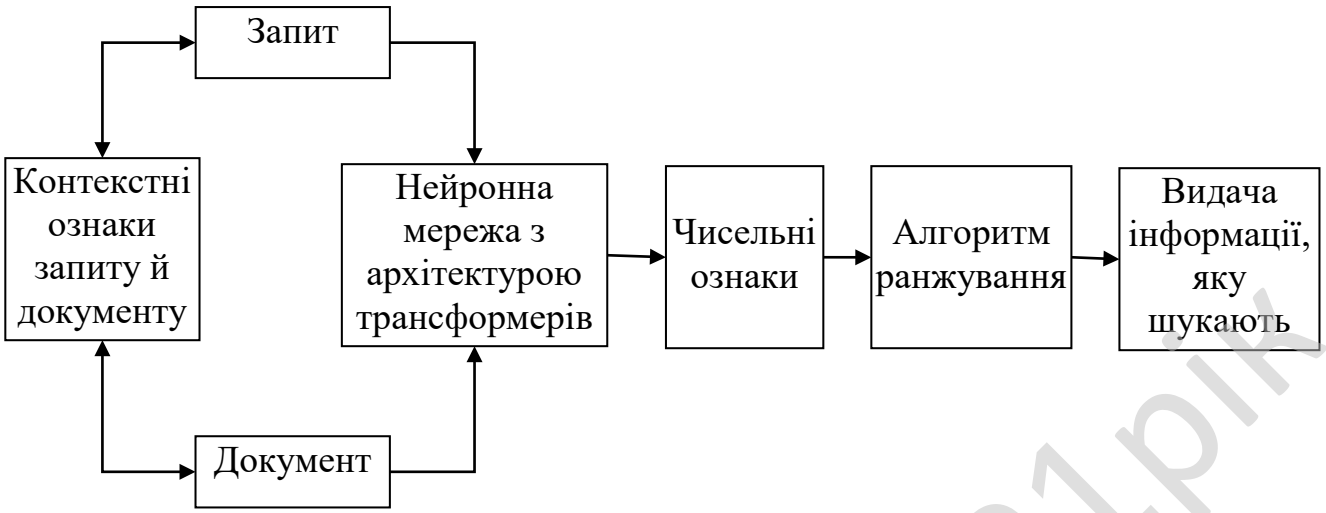

Рисунок 3.1 – Структурна схема системи

Нейромережі в ранжируванні

Сучасні пошукові системи розвиваються й поліпшують свою якість за рахунок усе більш точних наближень семантичному зв'язку запиту й документа, так що ілюзія розуміння стає повніше, охоплює нові класи запитів і користувацьких завдань. Основним інструментом для цього в останні роки стали нейронні мережі усе більш зростаючої складності. Втім, початок (як нам тепер видасться) було досить простим.

Перші нейронні мережі в пошуку мали просту feed-forward-архітектуру. На вхід мережі подається вихідний текст у вигляді «мішка слів» (bag of words). Кожне слово перетворюється у вектор, вектора потім підсумуються в один, який і використовується як вистава всього тексту. Взаємний порядок слів при цьому губиться або враховується лише частково за допомогою спеціальних технічних трюків. Крім того, розмір «словника» у такої мережі обмежений; невідоме слово в найкращому разі вдається розбити на частотні комбінації букв (наприклад, на триграмми) у надії зберегти хоча б частина його змісту. Вектор мішка слів потім пропускається через кілька щільних шарів нейронів, на виході яких утворюється семантичний вектор (інакше ембеддінг, від англ. embedding, вкладення; мається

на увазі, що вихідний об'єкт-текст вкладається в n-мірний простір семантичних векторів).

Чудова особливість такого вектора в тому, що він дозволяє наближати складні «значеннєві» властивості тексту за допомогою порівняно простих математичних операцій. Наприклад, щоб оцінити значеннєвий зв'язок запиту й документа, можна кожний з них спочатку перетворити в окремий вектор-ембеддінг, а потім вектора скалярно перемножити один на одного (або порахувати косинусну відстань).

У цій властивості семантичних векторів немає нічого дивного, адже нейронна мережа навчається вирішувати саме таке завдання, тобто наближати значеннєвий зв'язок між запитом і документом на мільярдах навчальних прикладів. У якості таргету (тобто цільового, дійсного значення релевантності) при цьому використовуються переваги наших користувачів, які можна визначити по балках пошуку. Точніше, можна припустити, що певний шаблон поведінки користувача добре корелює з наявністю (або, що не менш важливо, з відсутністю) значеннєвого зв'язку між запитом і показаним по ньому документом, і зібрати на його основі варіант таргету для навчання. Завдання складніше, чим може здатися на перший погляд, і допускає різні розв'язки. Якщо в якості позитивного прикладу звичайно підходить документ із «кліком» по запиту, то знайти гарний негативний приклад набагато сутужніше. Наприклад, виявляється, що майже даремно брати в якості негативних документи, для яких був показ, але не було «кліку» по запиту. Тут знову відкривається простір для евристичних алгоритмів, а їх правильне використання дозволяє на порядок поліпшити якість нейронної мережі, що виходить, у завданні ранжирування. Алгоритм, який найкраще показав себе на практиці, називається «алгоритмом переформулювань» і суттєво опирається на те, що користувачам властиво при розв'язку пошукового завдання задавати кілька запитів підряд, поступово уточнюючи формулювання, тобто «переформулювати» вихідний запит доти, поки не буде знайдений потрібний документ.

Важлива обмежуюча властивість такої мережі: увесь вхідний текст із самого початку представляється одним вектором обмеженого розміру, який повинен повністю описувати його «зміст». Однак реальний текст (в основному це стосується документа) має складну структуру – його розмір може сильно мінятися, а значеннєвий зміст може бути дуже різнорідним; кожне слово й пропозицію мають свій особливий контекст і додають свою частину змісту, при цьому різні частини тексту документа можуть бути в різному ступені пов'язані із запитом користувача. Тому проста нейронна мережа може дати лише дуже грубе наближення реальної семантики, яке в основному працює для коротких текстів.

Проте, використання щільних feed-forward-мереж у свій час дозволило суттєво поліпшити якість пошуку. У значній мірі цього вдалося добитися за рахунок тривалих експериментів з навчальними вибірками й добором правильних контентних ознак на вході моделей. Тобто ключовими питаннями для нас були: «На який (кліковий) таргет учити?» і «Які дані й у якому виді подавати на вхід?»

Цікаво, що хоча нейронна мережа навчається пророкувати зв'язок між запитом і документом, її потім можна легко адаптувати й для розв'язку інших значеннєвих завдань. Наприклад, уже готову мережу можна на порівняно невеликому числі прикладів донавчати для різних варіантів запитальної або документної класифікації, таких як виділення потоку порнозапитів, товарних і комерційних, навігаційних (тобто потребуючих конкретного сайту у відповіді) і так далі. Що ще більш важливо для ранжирування, таку ж мережу можна донавчати на експертних оцінках. Експертні оцінки в ранжируванні – це прямі оцінки значеннєвому зв'язку запиту й документа, які ставляться людьми на основі розуміння тексту. У складних випадках правильна оцінка вимагає спеціальних знань у вузькій області, наприклад у медицині, юриспруденції, програмуванні, будівництві. Це самий якісний і самий дорогою з доступних нам таргетів, тому важливо вміти його вивчати максимально ефективно.

Нейронна мережа, спочатку навчена на мільярдах «переформулювань», а потім донавчена на сильно меншій кількості експертних оцінок, помітно

поліпшує якість ранжирування. Це характерний приклад застосування підходу transfer learning: модель спочатку навчається вирішувати більш просту або більш спільне завдання на великій вибірці (цей етап також називають попереднім навчанням або переднавчанням, англ. pre-tain), а потім швидко адаптується під конкретне завдання вже на сильно меншому числі прикладів (цей етап називають донавчанням або налаштуванням, англ. fine-tune). У випадку простих feed-forward-мереж transfer learning уже приносить користь, але найбільшого ефекту цей метод досягає з появою архітектур наступного покоління.

Нейромережі-трансформери

Довгий час самої активною областю, що розвивається, застосування для складних алгоритмів аналізу тексту було завдання машинного перекладу. Результат роботи перекладача – це повністю згенерований текст, і навіть невеликі значеннєві помилки відразу помітні користувачеві. Тому для розв'язку завдань перекладу завжди використовувалися самі складні моделі, які могли врахувати порядок слів у тексті і їх взаємний вплив один на одного. Спочатку це були рекуррентні нейронні мережі (Recurrent Neural Networks, RNN), а потім трансформери. Далі мова йтиме тільки про трансформерах, які по суті є більш технологічно зробленим розвитком ідеї рекуррентних мереж.

У мережах з архітектурою трансформерів кожний елемент тексту обробляється окремо й представляється окремим вектором, зберігаючи при цьому своє положення. Елементом може бути окреме слово, знак пунктуації або частотна послідовність символів, наприклад BPE-токен. Мережа також включає механізм «уваги» (англ. attention), який дозволяє їй при обчисленнях «концентруватися» на різних фрагментах вхідного тексту.

Стосовно до завдання ранжирування, по запиту [купити кавоварку] така нейронна мережа може виділити частина документа (e.g. сторінки інтернет-магазину), у якій мова йде саме про потрібний користувачеві товарі. Інші частини теж можуть бути враховані, але їх вплив на результат буде менше. Трансформери можуть добре вивчати складні залежності між словами й активно

використовуються для генерації природніх текстів у таких завданнях, як переклад і відповіді на запитання (англ. question answering). Тому вони застосовуються вже досить давно. У першу чергу, звичайно, у Перекладачі.

У ранжируванні трансформери дозволяють добитися нового рівня якості при моделюванні семантичного зв'язку запиту й документа, а також дають можливість витягати корисну для пошуку інформацію з більш довгих текстів. Але одних тільки ванільних трансформерів для цього завдання мало.

Transformers + transfer learning

Глибокі нейронні мережі досить вимогливі до обсягу прикладів для навчання. Якщо даних мало, то ніякого виграшу від застосування важкої архітектури не вийде. При цьому практичних завдань завжди багато, і вони трохи відрізняються друг від друга. Зібрати мільярди прикладів для кожний завдання просто неможливо: не вистачить ні часу, ні бюджету. На допомогу знову приходить підхід transfer learning. Як ми вже розібралися на прикладі feed-forward-мереж, суть у тому, щоб перевикористовувати інформацію, накопичену в рамках одному завдання, для інших задач. цей підхід застосовується повсюдно, особливо він гарний у комп'ютерному зорі, де навчена на пошуку зображень базова модель легко донавчається майже на будь-які завдання. У трансформерах transfer learning теж очікуване заробив.

В 2018-м команда OpenAI показала, що якщо навчити трансформер на сирому корпусі текстів великого розміру в режимі язикової моделі, а потім дообучать модель на малих даних для конкретних завдань, то результат виявляється суттєво краще, чим раніше. Так народився проект GPT (Generative Pre-trained Transformer). Схожа ідея трохи пізніше лягла в основу проекту BERT (Bidirectional Encoder Representations from Transformers) від Google.

Такий же метод ми вирішили застосувати й у якості пошуку . Але для цього нам треба подолати кілька технологічних труднощів.

Трансформери в пошуку

Проект BERT чудовий тим, що кожний може взяти у відкритому доступі вже готову переднавчену модель і застосувати її до свого завдання. У багатьох випадках це дає відмінний результат. Але, на жаль, пошук – не такий випадок. Прості донавчені готові моделі приносить лише невелике поліпшення якості, зовсім непропорційне витратам ресурсів, які необхідні для застосування такої моделі в рантаймі. Щоб розкрити реальні можливості трансформера для пошуку, потрібно було навчати свою модель із нуля.

Для ілюстрації приведу кілька результатів з наших експериментів. Давайте візьмемо відкриту модель BERT-base Multilingual і навчимо на наші експертні оцінки. Можна виміряти корисність такого трансформера як додаткового фактора в завданні ранжирування; одержимо статистично значиме зменшення помилки пророкування релевантності на 3-4%. Це гарний фактор для ранжирування, який ми б негайно впровадили, якби він не вимагав застосування 12-слойного трансформера в рантаймі. Тепер візьмемо варіант моделі BERT-base, який ми навчили з нуля, і одержимо зменшення помилки пророкування майже на 10%, тобто більш ніж дворазовий ріст якості в порівнянні з ванільною версією, і це далеко не межа. Це не виходить, що модель Multilingual від Google – низької якості. За допомогою відкритих моделей BERT уже було отримано багато цікавих результатів у різних завданнях NLP (Natural Language Processing, тобто в завданнях обробки текстів природньою мовою). Але це значить, що вона погано підходить для ранжирування веб-сторінок на українській мові.

Перші труднощі, які виникає на шляху до навчання свого трансформера, – це обчислювальна складність завдання. Нові моделі добре масштабуються по якості, але при цьому в мільйони раз складніше, чим ті, які застосовувалися в пошуку раніше. Якщо раніше нейронну мережу вдавалося навчити за одну година, то трансформер на такому ж графічному прискорювачі Tesla v100 буде вчитися 10 років. Тобто без одночасного використання хоча б 100 прискорювачів (з можливістю швидкої передачі даних між ними) завдання не вирішується в

принципі. Необхідний запуск спеціалізованого обчислювального кластера й розподілене навчання на ньому.

Нам треба було небагато часу, пари сотень GPU-карт, місце в одному з дата-центрів і класні інженери. Ми зібрали кілька версій кластера й успішно запустили на ньому навчання. Тепер модель одночасно навчається приблизно на 100 прискорювачах, які фізично розташовані в різних серверах і спілкуються один з одним через мережу. І навіть із такими ресурсами на навчання йде близько місяця.

Кілька слів про саме навчання. Нам важливо, щоб модель, що вийшов, вирішувала з оптимальною якістю саме завдання ранжирування. Для цього ми розробили свій стек навчання. Як і BERT, модель спочатку вчиться властивостям мови, вирішуючи завдання MLM (Masked Language Model), але робить це відразу на текстах, характерних для завдання ранжирування. Уже на цьому етапі вхід моделі складається із запиту й документа, і ми із самого початку навчаємо модель пророкувати ще й імовірність кліку на документ по запиту. Дивно, але той же самий таргет «переформулювань», який був розроблений ще для feed-forward-мереж, відмінно показує себе й тут. Навчання на клік суттєво збільшує якість при наступному розв'язку семантичних завдань ранжирування.

У донавчені ми використовуємо не одне завдання, а послідовність завдань зростаючої складності. Спочатку модель вчиться на більш простих і дешевих толокерских оцінках релевантності, якими ми розташовуємо у великій кількості. Потім на більш складних і дорогих оцінках асессорів. І нарешті, навчається на підсумкову метрику, яка поєднує в собі відразу кілька аспектів і по якій ми оцінюємо якість ранжирування. Такий метод послідовного донавчення дозволяє добитися найкращого результату. По суті весь процес навчання вибудовується від більших вибірок до малих і від простих завдань до більш складних і «семантичних».

На вхід моделі ми подаємо всі ті ж контентні ознаки, про яких ішла мова на самому початку. Тут є текст запиту, його розширення й фрагменти вмісту

документа. Документ цілком усе ще занадто великий, щоб модель могла впоратися з ним повністю сама, без попередньої обробки. Стріми теж містять корисну інформацію, але з ними нам ще треба попрацювати, щоб зрозуміти, як її ефективно донести до пошуку. Складніше всього виявилося виділити гарні фрагменти тексту документа. Це цілком очікуване: щоб виділити найбільш важливу частину веб-сторінки, уже потрібно багато чого розуміти про її значеннєвий зміст. Зараз нам на допомогу знову приходять прості евристики й алгоритми автоматичної сегментації, тобто розмітки сторінки на структурні зони (основний зміст, заголовки і так далі). Вартість помилки досить велика, іноді для правильного розв'язку завдання необхідно, щоб модель «побачила» одна конкретну пропозицію в довгому тексті статті, і тут ще залишається потенціал для поліпшень.

Отже, ми навчилися навчати моделі в офлайні, але от працювати їм уже потрібно в онлайні, тобто в реальному часі у відповідь на тисячі користувацьких запитів у секунду. Отут полягає другі принципові труднощі. Застосування трансформера – це важке для рантайма завдання; моделі такої складності можна застосовувати тільки на GPU-картах, інакше час їх роботи виявиться надмірним і легко може перевищити час роботи всього пошуку. Треба було розробити з нуля й розгорнути кілька сервісів для швидкого застосування («інференсу», англ. inference) трансформерів на GPU. Це новий тип інфраструктури, який до цього не використовувався в пошуку.

Безпосередньо для застосування ми доробили внутрішню бібліотеку для інференсу трансформерів, яка розроблена нашими колегами з Яндекс.Перекладача й, по наших вимірах, як мінімум не уступає іншим доступним аналогам. Ну й звичайно, усе вважається в FP16 (16-бітна вистава чисел із плаваючою крапкою).

Однак, навіть із урахуванням використання GPU і оптимізованого коду для інференсу, модель із максимальним рівнем якості занадто більша для впровадження в рантайм пошуку. Для розв'язку цієї проблеми є класичне

приймання – knowledge distillation (або dark knowledge). ми використовуємо менш пафосний термін «пародіювання». Це навчання більш простої моделі, яка «пародіює» поведінка більш складної, навчаючись на її пророкування в офлайні.

У результаті пародіювання складність моделі вдається зменшити в рази, а втрати якості залишаються в межах ~10%.

Це не єдиний спосіб оптимізації. Ми також навчилися робити багатозадачні моделі, коли одна модель « наслідує» відразу декількома складним моделям, навченим на різні завдання. При цьому сумарна корисність моделі для ранжирування суттєво зростає, а її складність майже не збільшується.

Якщо ж хочеться добитися максимальної якості, то можливостей однієї дистиляції недостатньо. Доводиться поділити модель на кілька частин, кожна з яких застосовується незалежно. Частина моделі застосовується тільки до запиту; частина – тільки до документа; а то, що вийшло, потім обробляється фінальною єднальною моделлю. Такий метод побудови «роздільної» або split-моделі розподіляє обчислювальне навантаження між різними компонентами системи, що дозволяє зробити модель більш складної, побільшати розмір її входу й помітно підвищити якість.

Впровадження split-моделі знову приводить нас до нових і цікавих інженерних завдань, але розповісти про все в одному пості, на жаль, неможливо. Хоча архітектура нейромереж-трансформерів відома вже досить давно, а їх використання для завдань NLP набуло величезну популярності після появи BERT в 2018 році, впровадження трансформера в сучасну пошукову систему неможливо без інженерної винахідливості й великого числа оригінальних технологічних поліпшень у навчанні й рантаймі. Тому називається технологія YATI – Yet Another Transformer (with Improvements), що, як нам здається, добре відбиває її суть. Це дійсно «ще один трансформер», архітектурно схожий на інші моделі (а їх в останні роки з'явилося безліч), але унікальний тим, що завдяки сукупності поліпшень він здатний працювати й приносити користь у пошуку – самому складному сервісі .

Що в підсумку? У нас з'явилася своя інфраструктура навчання й дистиляції важких моделей, адаптована під наш стек завдань ранжирування. З її допомогою ми спочатку навчили більші моделі-трансформери високої якості, а потім дистилювали їх у багатозадачну split-модель, яка впроваджена в рантайм на GPU у вигляді декількох частин, незалежно застосовуваних до запиту й документу.

3.3 Розробка функціональної схеми

У даній роботі реалізований наступний функціонал:

Керування змістом системи користувацького пошуку:

– Додавання нових сайтів або сторінок, що збільшують охват системи користувацького пошуку.

– Відправлення запитів на сканування й індексування нових сторінок пошукової системи.

Розширення можливостей пошуку:

– Розподіл результатів пошуку по категоріях, завдяки чому користувачі зможуть точніше вказувати потрібні їм дані.

– Додавання у верхній частині результатів пошуку вікна з URL, на який ви прагнете звернути увагу користувачів.

– Розширення можливостей пошуку за рахунок автоматичного включення в пошуковий запит синонімів ключових слів. Наприклад, ви можете вказати, щоб для кожного пошуку по слову [програміст] верталися також результати за словами [програміст на java].

– Економія часу користувачів за рахунок відображення результатів по схожих запитах.

– Зміна зовнішнього вигляду пошукової системи.

Рисунок 3.2 – Функціональна схема системи

Заробіток:

– Прив'язування до безкоштовної СПП аккаунту Adsense.

Керування системою користувацького пошуку:

– Зміна назви пошукової системи, визначення її ідентифікатора й зміна основних налаштувань (наприклад, мови).

– Додавання адміністраторів у Систему користувацького пошуку. Вони зможуть додавати сайти, уточнення й просування результатів пошуку, але не зможуть додавати інших користувачів або відкривати сторінку "Заробіть!".

– Перегляд історії змін.

– Тестування Системи користувацького пошуку.

– Статистика використання Системи користувацького пошуку.

– Відновлення фрагмента коду на сайті. Це необхідно робити при внесенні будь-яких змін у пошукову систему.

3.4 Розробка діаграми процесів

Відповідно до методичних рекомендацій розроблення графічної частини кваліфікаційної бакалаврської роботи розглянемо розроблену діаграму процесів яка зображена на рисунку 3.3.

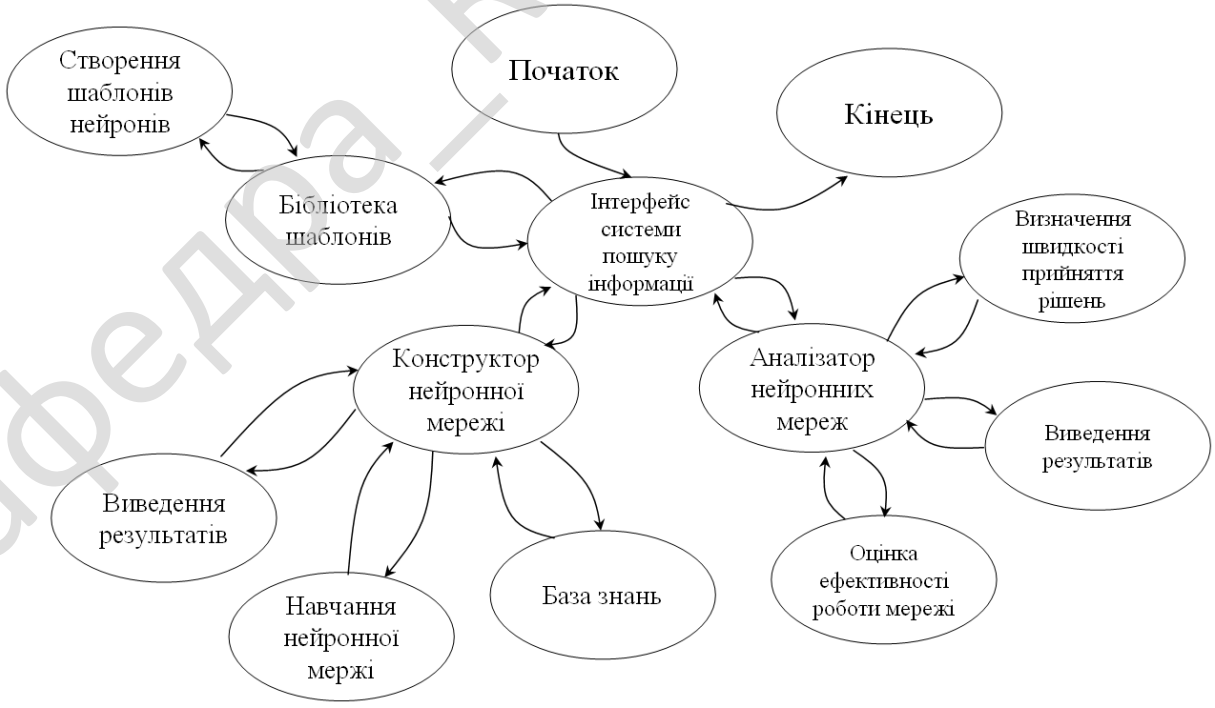

Рисунок 3.3 – Діаграма взаємодії процесів

Розроблена діаграма взаємодії процесів використовується для представлення та візуалізації процесів обробки даних тобто структурного проектування бакалаврської роботи.

Відповідно до документації основна будова діаграми процесів полягає у графічному представленні складу сукупностей даних, що характеризуються як співвідношення різних частин кожної з сукупностей.

Склад статистичної сукупності графічно може бути представлений як за допомогою абсолютних, так і відносних показників. Графічне зображення складу сукупності по абсолютними і відносними показниками сприяє проведенню більш глибокого аналізу і дозволяє проводити аналіз системи.

Основні складові елементи діаграми взаємодії процесів це потоки даних:

– Репозиторії, потік сховища даних.

– Потоки зовнішні по відношенню до системи сутності.

– Процеси які являють собою трансформацію даних в рамках описуваної системи.

– Потоки даних гібридні між елементами трьох попередніх типів.

Для схематичного представлення системи що розробляється необхідно спочатку представити діаграму взаємодії процесів даних рівня контексту, завдяки чому буде показано взаємодію системи в цілому у подальшому.

Використовується модель проектування, графічне представлення «потоків» даних в інформаційній системі. Розроблена діаграма взаємодії процесів системи в подальшому уточнюється шляхом деталізації процесів та потоків даних з метою показати систему що розробляється.

Таким чином у результаті після розгляду, вищеописаної системи, схеми структурної, функціональної, діаграми взаємодії процесів перейдемо до опису та розгляду блок-схем основної програми, та підпрограм, які використовуються, для реалізації системи.

4 РЕАЛІЗАЦІЯ РОБОТИ. РОЗРАХУНКИ І ЕКСПЕРИМЕНТАЛЬНІ ДАНІ, ЩО ПІДТВЕРДЖУЮТЬ ВІРНІСТЬ ПРОЕКТНИХ ТА ПРОГРАМНИХ РІШЕНЬ

4.1 Блок-схеми та опис алгоритмів функціонування системи

У розділі розглянемо реалізацію бакалаврської дипломної роботи та наведені приклади блок-схем та опис алгоритмів функціонування системи у вигляді частин вихідного коду.

Розглянемо частину реалізованого коду який призначено для початкового пошуку інформації без застосування технології нейронних мереж з архітектурою трансформерів що було застосовано. На цьому модулі проходе порівняння отриманих даних та формується звіт якості пошуку з використанням запропонований нейроних підходів та без цього підходу. Вихідний код наступний.

```
unit Unit5;

// інтерфейс модуля
interface

// Підключення бібліотек
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
Dialogs, StdCtrls,ShellAPI, Menus;

// опис типів
type
  TForm5 = class(TForm)
    Edit1: TEdit;
    GroupBox1: TGroupBox;
    Button1: TButton;
    PopupMenu1: TPopupMenu;
    Label1: TLabel;
    Button2: TButton;
    Yandexru1: TMenuItem;
    N1: TMenuItem;
```

```
    MSN1: TMenuItem;
    Yahoo1: TMenuItem;
    Rambler1: TMenuItem;
    ru1: TMenuItem;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure N1Click(Sender: TObject);
    procedure MSN1Click(Sender: TObject);
    procedure Yahoo1Click(Sender: TObject);
    procedure Rambler1Click(Sender: TObject);
    procedure ru1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
  private
    { Private declarations }
  procedure CreateParams(var Params: TCreateParams); override;
  public
    { Public declarations }
  end;

// підключення типів та створення змінних
var
  Form1: TForm5;
  estyle:integer;
  poisk:string;
// змінна пошукача даних

implementation
// Реалізація інтерфейсної частини
{$R *.dfm} // ресурси

//Функция запуска
function ExecuteFile(const FileName,Params,DefaultDir:string;
ShowCmd:Integer):THandle;
var
zFileName,zParams,zDir:array [0..79 ] of Char;
begin
Result :=ShellExecute(Application.MainForm.Handle,nil,
StrPCopy(zFileName,FileName),StrpCopy(zParams,Params),
StrPCopy(zDir,DefaultDir),ShowCmd);
end;
```

```
procedure TForm5.CreateParams(var Params: TCreateParams);
begin
// видаляємо форму
inherited CreateParams(Params);
Params.Style := Params.Style or ws_popup xor ws_dlgframe;
end;

// створення форми
procedure TForm5.FormCreate(Sender: TObject);
var FormRgn: hRgn;
begin
estyle:=getwindowlong(application.Handle,gwl_exstyle);
setwindowlong(application.Handle,gwl_exstyle,estyle or ws_ex_toolwindow);
ScreenSnap:=TRUE;
SnapBuffer:=60;
Form1.Brush.Style := bsSolid;
GetWindowRgn(Form1.Handle, FormRgn);
DeleteObject(FormRgn);
Form1.Height := 50;
Form1.Width := 346;
SetWindowRgn(Form1.Handle, FormRgn, TRUE);
end;

procedure TForm5.Button1Click(Sender: TObject);
var s,sear:string;c,i:integer;mass:array[1..30] of string;
begin
// Присвоюємо фразу яку будемо шукати
s:=edit1.Text;
// Знаходимо скільки символів в рядку
c:=Length(s);
// цікл який шукає прогалини в фразі і замість пробілу ставить +
for i:=1 to c do
begin
mass[i]:=copy(s,i,1);
if mass[i]=' ' then
mass[i]:='+';
sear:=sear+mass[i];
end;
// Відкриваємо і шукаємо
ExecuteFile(poisk+sear,'','',SW_SHOW);
end;
```

```
procedure TForm5.N1Click(Sender: TObject);
begin
poisk:='http://www.google.com/search?hl=ru&q=';
label1.Caption:='Пошук на Google.com'
end;

procedure TForm5.MSN1Click(Sender: TObject);
begin
poisk:='http://search.msn.com/results.aspx?q=';
label1.Caption:='Пошук на MSN.com'
end;

procedure TForm5.Yahoo1Click(Sender: TObject);
begin
poisk:='http://search.yahoo.com/bin/query?p=';
label1.Caption:='Пошук на Yahoo.com'
end;

procedure TForm5.Button2Click(Sender: TObject);
begin
 PopupMenu1.Popup(Mouse.CursorPos.X,Mouse.CursorPos.Y);
end;

procedure TForm5.Button3Click(Sender: TObject);
begin
 Close;
end;
end.
```

Були проведені розрахунки і підібрані набори тестових даних для перевірки правильності реалізації проектних рішень. Було створено блок-схеми роботи системи. Блок-схеми показують весь процес роботи системи з підсистемами та частково доказують правильність вибраних проектних рішень. Тому від точності і детальної блок-схеми залежить результат всієї програми.

При виборі початкової точки відліку при побудові схем було враховано, що виходячи з вибору мови програмування і інших технічних засобів, програма буде об’єктно-орієнтована що вимагає високого рівня декомпозиції задач на класи.

На рисунку 4.1 зображена основна блок-схема програми, на рисунку 4.2 зображено роботу підпрограми.

З яких видно що робота основної програми складається з початкових етапів ініціалізації ПЗ, перевірки наявності ресурсів системи, блоку початку основного циклу з чеканням запиту від користувача в якому відбувається виклик підсистеми та останньої стадії – перевірка поточного стану з завершенням роботи розробленого ПЗ.

При роботі підпрограми виконується основний функціонал системи з циклічними послідовностями, перевіркою поточного стану та поверненням в основну програму прапорів стану виконання.

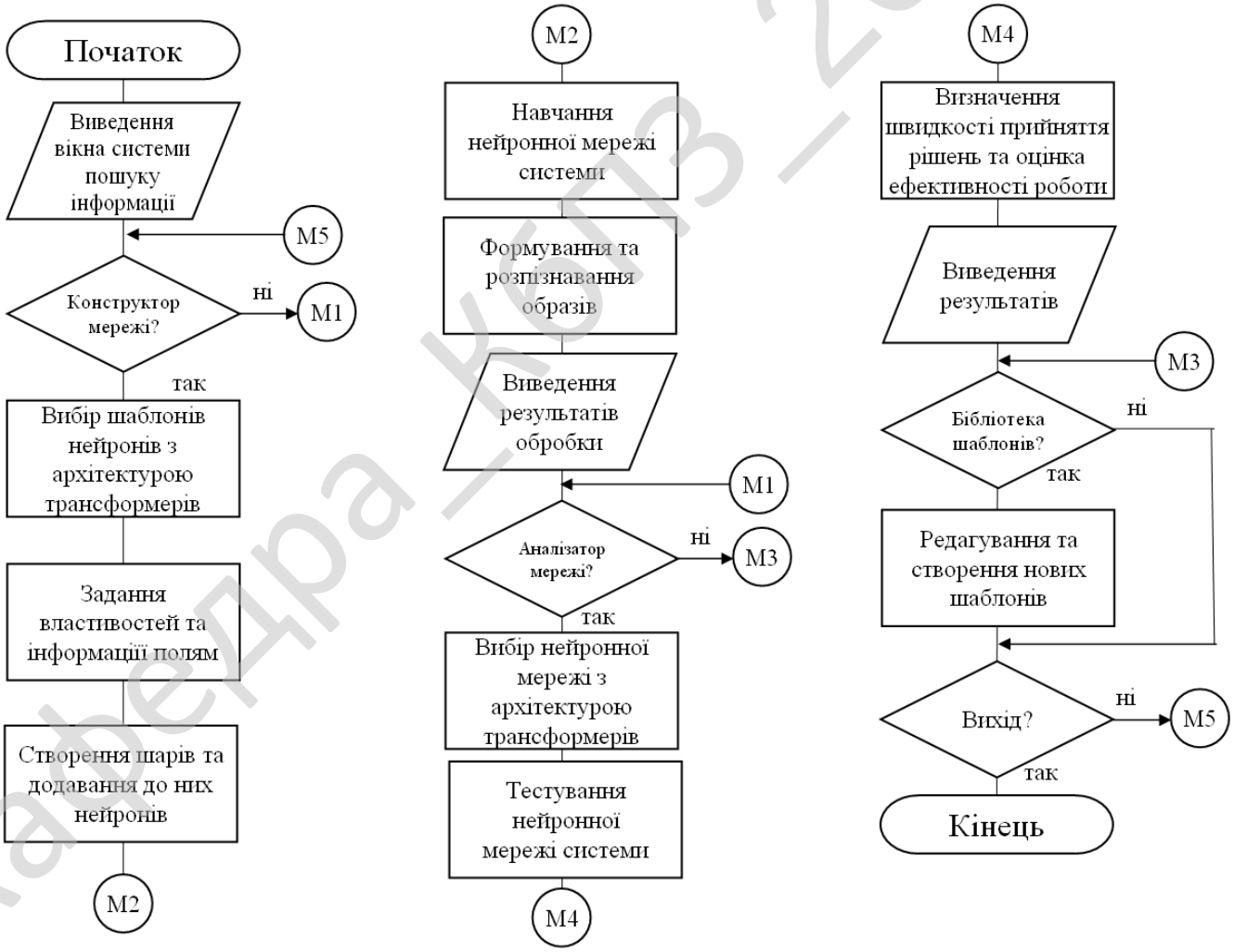

Рисунок 4.1 – Блок-схема основної програми

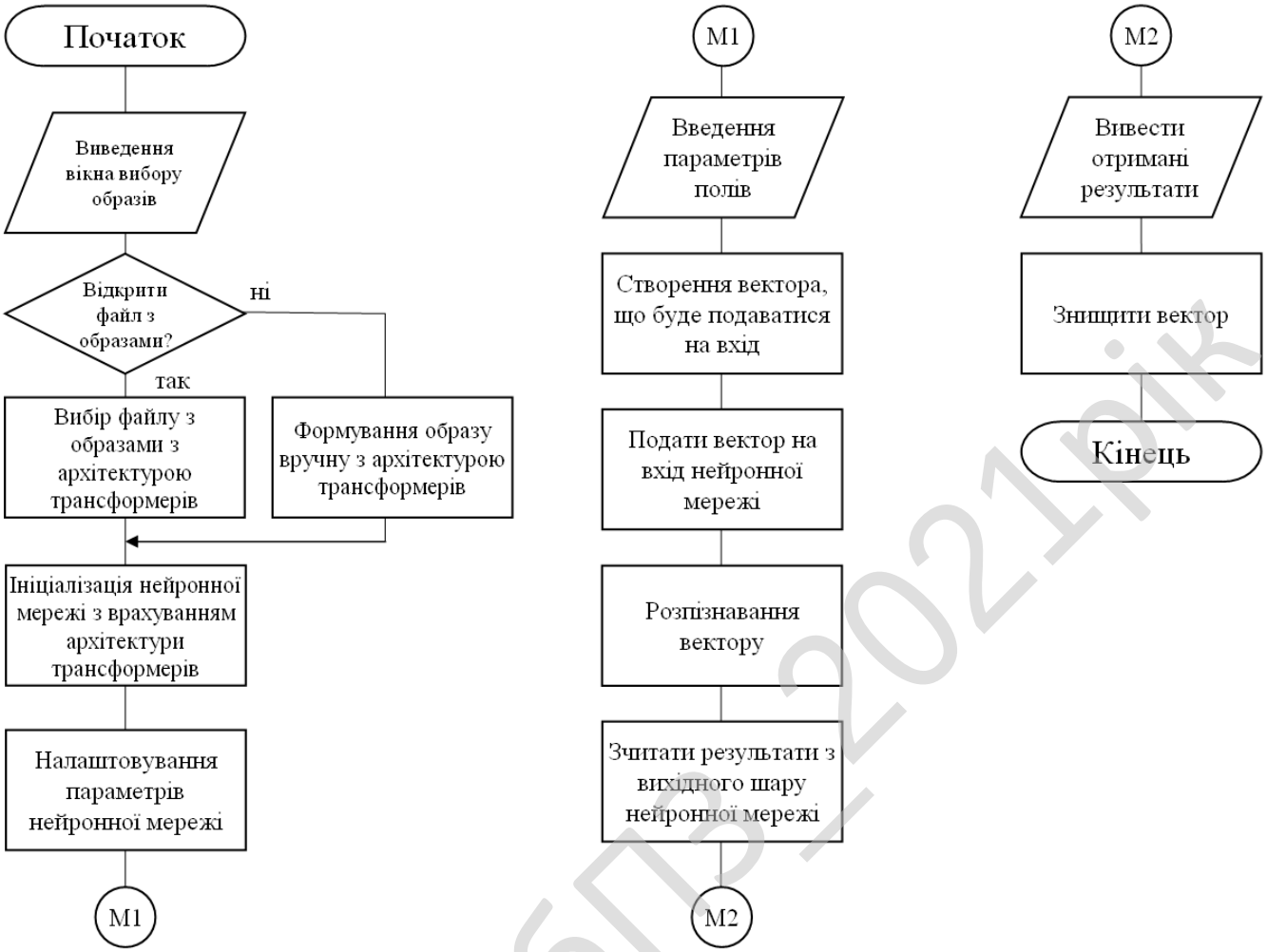

Рисунок 4.2 – Блок-схема роботи підпрограми

Розглянемо формат що було використано – JSON (JavaScript Object Notation, укр. запис об'єктів JavaScript, вимовляється джейсон) – це текстовий формат обміну даними між комп'ютерами.

JSON базується на тексті, може бути прочитаним людиною. Формат дозволяє описувати об'єкти та інші структури даних. Цей формат головним чином використовується для передачі структурованої інформації через мережу (завдяки процесу, що називають серіалізацією). Розробив і популяризував формат Дуглас Крокфорд.

JSON знайшов своє головне призначення у написанні веб-програм, а саме при використанні технології AJAX. JSON, що використовується в AJAX, виступає

як заміна XML (використовується в AJAX) під час асинхронної передачі структурованої інформації між клієнтом та сервером. При цьому перевагою JSON перед XML є те, що він дозволяє складні структури в атрибутах, займає менше місця і прямо інтерпретується за допомогою JavaScript в об'єкти.

JSON з'явився через необхідність обміну даними з сервером у реальному часі без використання плагінів для браузерів, flash-додатків або Java-аплетів, які використовувались скрізь на початку 2000-х років. Дуглас Крокфорд був тим, хто активно просував новий на той час формат. Він з колегами хотів створити технологію, яка використовувала б можливості звичайного браузера давала б веб-розробникам можливість створювати веб-застосунки із постійним двостороннім зв'язком із веб-сервером.

JSON вперше був використаний в проекті в Communities.com для Cartoon Network, він дозволяв обмінюватись повідомленнями і одночасно маніпулювати DHTML-елементами.

Веб-сайт JSON.org було запущено 2002 року. У грудні 2005-го року Yahoo! почав переводити деякі зі своїх веб-сервісів на роботу з JSON. Google взялася до роботи з технологією у своєму веб-протоколі GData у грудні 2006-го року.

Використання

За рахунок своєї лаконічності в порівнянні з XML, формат JSON може бути більш придатним для серіалізації складних структур.

Якщо говорити про веб-застосунки, в такому ключі він доречний в задачах обміну даними як між браузером і сервером (AJAX), так і між самими серверами (програмні HTTP-інтерфейси).

Формат JSON так само добре підходить для зберігання складних динамічних структур в реляційних базах даних або файловому кеші.

Синтаксис

JSON будується на двох структурах:

1. Набір пар назва/значення. У різних мовах це реалізовано як об'єкт, запис, структура, словник, хеш-таблиця, список з ключем або асоціативним масивом.

2. Впорядкований список значень. У багатьох мовах це реалізовано як масив, вектор, список, або послідовність.

Це універсальні структури даних. Теоретично всі сучасні мови програмування підтримують їх у тій чи іншій формі. Оскільки JSON використовується для обміну даними між різними мовами програмування, то є сенс будувати його на цих структурах.

У JSON використовуються такі їхні форми:

1. Об'єкт – це послідовність пар назва/значення. Об'єкт починається з символу { і закінчується символом } . Кожне значення слідує за : і пари назва/значення відділяються комами.

2. Масив – це послідовність значень. Масив починається символом [і закінчується символом]. Значення відділяються комами.

3. Значення може бути рядком в подвійних лапках, або числом, або логічними true чи false, або null, або об'єктом, або масивом. Ці структури можуть бути вкладені одна в одну.

4. Рядок – це послідовність з нуля або більше символів юнікода, обмежена подвійними лапками, з використанням escape-послідовностей, що починаються зі зворотної косої риски \. Символи представляються простим рядком. Тип Рядок (String) дуже схожий на String в мовах C і Java. Число теж дуже схоже на C- або Java-число, за винятком того, що вісімкові та шістнадцяткові формати не використовуються. Пропуски можуть бути вставлені між будь-якими двома лексемами.

Використання JSON в AJAX

Наступний фрагмент коду JavaScript показує, як клієнт може використати XMLHttpRequest для запиту об'єктів в форматі JSON з серверу. (Серверна частина коду пропущена, вона просто повертає на запит URL рядок у JSON форматі).

Треба відзначити, що тут використання XMLHttpRequest не є кросс-браузерним (за деталями звертайтеся на сторінку XMLHttpRequest), і код зазнаватиме незначних модифікацій в різних версіях оглядачів Internet Explorer, Opera, Safari або Mozilla. Використання запиту XMLHttpRequest обмежено правилом одного джерела (same origin policy): URL, що відповідає на запит, має посилатися на той же сайт, що обслуговує сторінку, що ініціювала запит.

Оглядачі можуть також використовувати тег <iframe> для асинхронного запиту JSON-даних в кросс-браузерному варіанті, або використати просте перенаправлення <form action="url_to_cgi_script" target="name_of_hidden_iframe">. Такий підхід був поширений до приходу популярного нині запиту XMLHttpRequest.

Динамічний тег <script> також можна використати для підвантаження JSON-даних. Ця техніка можлива, щоб обійти надто суворе правило одного джерела, але вона не є безпечною. Запит JSONRequest пропонується, як безпечніша альтернатива.

Питання безпеки

Хоча JSON призначений для передачі даних в серіалізованому вигляді, його синтаксис відповідає синтаксису JavaScript і це створює низку проблем безпеки. Часто для обробки даних, отриманих від зовнішнього джерела у форматі JSON, до них застосовується функція eval() без якої-небудь попередньої перевірки.

JavaScript eval()

Оскільки JSON представляється синтаксично правильним фрагментом коду JavaScript, природним способом розбору JSON-даних в JavaScript-програмі є використання вбудованої в JavaScript функції eval(), яка призначена для обчислення JavaScript-виразів. При цьому підході відпадає необхідність у використанні додаткових парсерів.

Техніка використання eval() робить систему вразливою, якщо джерело JSON-даних, що використовуються, не відноситься до надійних. Такими даними

може виступати шкідливий JavaScript-код для атак за допомогою ін'єкції коду. За допомогою цієї вразливості можливо здійснювати крадіжку даних, підробку автентифікації. Проте, вразливість можна усунути за рахунок використання додаткових засобів перевірки даних на коректність. Наприклад, до виконання eval() отримані від зовнішнього джерела дані можуть перевірятися за допомогою регулярних виразів.

Вбудований JSON

Останні версії веб-браузерів мають вбудовану підтримку JSON і здатні його обробляти без виклику функції eval(), що призводить до описаної проблеми. Обробка JSON у такому разі зазвичай здійснюється швидше. Так у червні 2009 року вбудовану підтримку JSON мали такі браузери: Mozilla Firefox 3.5+; SeaMonkey 2; Thunderbird 3; Microsoft Internet Explorer 8; Opera 10.5; Браузери, засновані на WebKit (наприклад, Google Chrome, Apple Safari).

Принаймні дві популярні бібліотеки JavaScript використовують вбудований JSON у разі його доступності: jQuery; Dojo.

Підробка крос-доменного запиту

Непродумане використання JSON робить сайти вразливими до підробки міжсайтових запитів (CSRF або XSRF). Оскільки тег <script> допускає використання джерела, що не належить до того ж домену, що і використовуваний ресурс, це дозволяє виконувати код даних, представлених у форматі JSON, в контексті довільної сторінки, що робить можливою компрометацію паролів або іншої конфіденційної інформації користувачів, що пройшли авторизацію на іншому сайті.

Це є проблемою тільки у разі вмісту в JSON-даних конфіденційної інформації, яка може бути компрометована третьою стороною і якщо сервер розраховує на політику одного джерела, блокуючи доступ до даних при виявленні зовнішнього запиту. Це не є проблемою, якщо сервер визначає допустимість запиту, надаючи дані тільки у разі його коректності. HTTP cookie не можна

використовувати для визначення цього. Виключне використання HTTP cookie використовується підробкою міжсайтових запитів.

Розширення, JSONP

JSONP або «JSON з підкладкою» є розширенням JSON, коли назва функції зворотного виклику вказується як вхідний аргумент. Спочатку ідея була запропонована в блозі MacPython в 2005 році, і в наш час використовується багатьма Web 2.0 застосунками, такими, як Dojo Toolkit Applications, Google Toolkit Applications і zanox Web Services.

Подальші розширення цього протоколу були запропоновані з урахуванням введення додаткових аргументів, як, наприклад, у разі JSONPP за підтримки S3DB вебсервісів.

Оскільки JSONP використовує скрипт-теги, виклики по суті відкриті світові. З цієї причини JSONP може бути недоречними для зберігання конфіденційних даних.

Включення скриптових тегів від віддалених сайтів дозволяє їм передати будь-який контент на сайті. Якщо віддалений сайт має вразливості, які дозволяють виконати ін'єкції JavaScript, то початковий сайт також може бути ними зачеплений.

Розширення, BSON

BSON – це бінарна форма представлення простих структур даних і асоціативних масивів (які називають об'єктами або документами). Назва «BSON» заснована на визначенні JSON і неофіційно означає «Binary JSON» (бінарний JSON).

JSON Reference

Стандарт JSON не описує посилання на інші об'єкти або частини, але існує чернетка стандарту IETF для посилань на об'єкти на основі JSON. Посилання дозволяють здійснювати трансклюзію – вставляти одні документи в інші.

4.2 Захист розробленого програмного забезпечення

Розроблене програмне забезпечення захистимо за допомогою алгоритму захисту інформації RSA. Спочатку необхідно обчислити пару ключів (секретний ключ і відкритий ключ). Для цього відправник електронних документів обчислює два більших простих числа P и Q, потім знаходить їхній добуток $N = P*Q$ і значення функції $\varphi(N) = (P-1)(Q-1)$. Далі відправник обчислює число E з умов $E < \varphi(N)$, НЗД $(E, \varphi(N)) = 1$ і число D з умов $D < N$, $E*D \equiv 1 (\text{mod } \varphi(N))$.

Пари чисел (E, N) є відкритим ключем. Цю пару чисел автор передає партнерам по переписці для перевірки його цифрових підписів. Число D зберігається автором як секретний ключ для підписування.

Допустимо, що відправник хоче підписати повідомлення M перед його відправленням. Спочатку повідомлення M (блок інформації, файл, таблиця) стискають за допомогою геш-функції h(-) у ціле число m: $m = h(M)$.

Потім обчислюють цифровий підпис S під електронним документом M, використовуючи геш-значення m і секретний ключ D: $S = m \pmod N$.

Пари (M, S) передається партнерові-одержувачеві як електронний документ M, підписаний цифровим підписом S, причому підпис S сформований власником секретного ключа D.

Після прийому пари (M, S) одержувач обчислює геш-значення повідомлення M двома різними способами. Насамперед, він відновлює геш-значення m', застосовуючи криптографічне перетворення підпису S з використанням відкритого ключа E: $m' = S^E \pmod N$.

Крім того, він знаходить результат гешування прийнятого повідомлення M с допомогою такої ж геш-функції h(-): $m = h(M)$.

Якщо дотримується рівність обчислених значень, тобто $S^E(\text{mod}) = h(M)$, то одержувач визнає пару (M, S) справжньою.

Доведено, що тільки власник секретного ключа *D* може сформувати цифровий підпис *S* по документі *M*, а визначити секретне число *D* по відкритому числу *E* не легше, ніж розкласти модуль *N* на множники.

Крім того, можна строго математично довести, що результат перевірки цифрового підпису *S* буде позитивним тільки в тому випадку, якщо при обчисленні *S* був використаний секретний ключ *D*, що відповідає відкритому ключу *E*.

Тому відкритий ключ E іноді називають "ідентифікатором" того, хто підписав.

5 МЕТОДИКА ВПРОВАДЖЕННЯ СИСТЕМИ В ПРОМИСЛОВУ ЕКСПЛУАТАЦІЮ

Програма має простий та інтуїтивно зрозумілий інтерфейс, який зображений на рисунку 5.1. З рисунку головного вікна можна побачити що інтерфейс головного вікна розподілено на наступні функціональні розділи:

– Функціональних кнопок ПЗ: Навчання; Налаштування; Додавання правил; Видалення правил. Які присутні у відповідних розділах підгруп

– Верхнього меню: Файл; Конструктор; Аналізатор; Довідка.

– Розділу обрання підгрупи вікна: Створення нейромережі; Навчання; Розпізнавання; База знань.

– Навігаційного меню яке викликається натисканням правої клавіші маніпулятора миші: Налаштування; Журнал роботи.

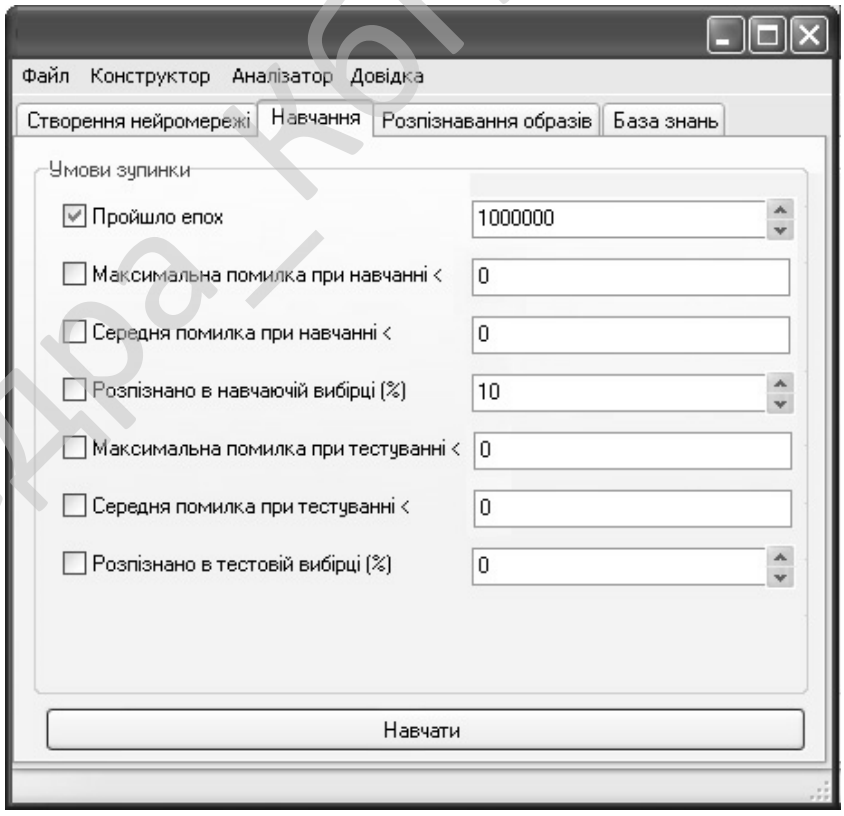

Рисунок 5.1 – Головне вікно ПЗ

На рисунку 5.2 зображено авторські дані розробленого програмного забезпечення. Розроблена програма має дуже простий і зрозумілий інтерфейс з користувачем. Кожен, хто в достатньому обсязі володіє операційним середовищем Windows без особливих складностей освоїть і цю програму, оскільки її інтерфейс інтуїтивно зрозумілий. Якщо програма не видала ніяких помилок, і працює, то можна використовувати, інакше слід слідувати інструкціям, які пропонує програма.

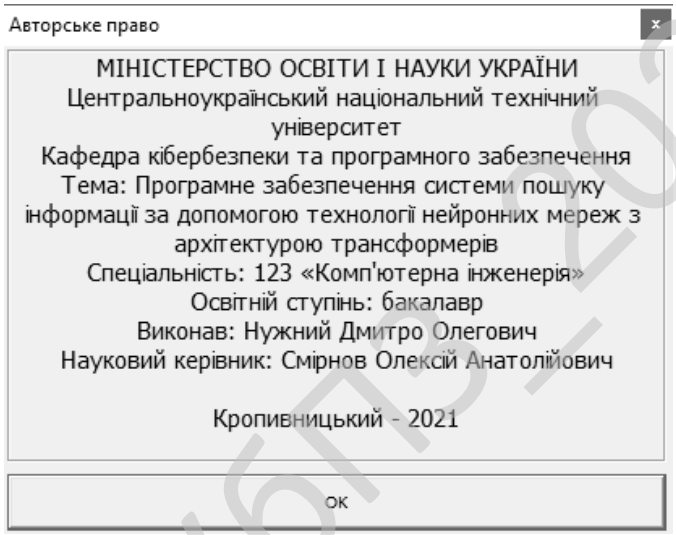

Рисунок 5.2 – Авторське право

Процес впровадження програмного забезпечення, це процес налаштування програмного забезпечення під певні умови використання, а також навчання користувачів роботі з програмним продуктом. Впровадження програмного забезпечення це усі дії, що роблять розроблену програмну систему готовою до використання. Даний процес є частинною життєвого циклу програмного забезпечення. Таким чином у результаті вищерозглянутого можна стверджувати що розроблено інтерфейс системи у відповідності з вибраною метою роботи. Система містить максимальний необхідний набір функцій придатних для виконання будь-яких дій для забезпечення повноцінної роботи програми. Далі розглянемо висновки та використані літературні джерела.

6 ОСНОВНІ ВИСНОВКИ

Програмне забезпечення, створене в результаті виконання кваліфікаційної бакалаврської роботи, призначено для системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

В межах України в недостатній мірі представлені вітчизняні розробки в цій області.

Рішення завдання полягало у вирішенні наступних задач:

– Був проведений огляд існуючих систем пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

– Досліджена система пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

– На основі отриманих результатів досліджень створена програмна реалізація системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

Розроблені під час виконання кваліфікаційної бакалаврської роботи алгоритми дозволяють успішно вирішувати завдання пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

Розроблене програмне забезпечення має простий, дружній та зручний інтерфейс користувача, що забезпечує легкість у освоєнні роботи програмного продукту, зручність у використанні, і не потребує особливих спеціальних знань.

При створені програмного забезпечення було використано об’єктно-орієнтований підхід, що відповідає сучасним тенденціям у галузі розробки комерційних програмних систем.

Програма реалізована на мові високого рівня RAD Studio Delphi 10.4. Дана мова програмування дозволяє найбільш ефективно обробляти дані призначені для системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів. Це дозволило мінімізувати строк розробки

програмного забезпечення, і, як слід, зменшити витрати на його розробку. Запропоноване програмне забезпечення ділиться на загальне програмне забезпечення, що поставляється із засобами обчислювальної техніки й спеціальне програмне забезпечення, що спеціально розроблене для даної конкретної системи й включає програми, що реалізують її функції.

Програма призначена для виконання під управлінням багатозадачної операційної системи Windows XP/Vista/7/8/10.

Даються необхідні рекомендації з установки розробленого програмного забезпечення.

Для підвищення рівня безпеки запропоновано застосовувати алгоритм RSA.

В цілому створене програмне забезпечення підтверджує правильність використаних проектних рішень та повністю відповідає вимогам технічного завдання. Створене програмне забезпечення має потенційну можливість для подальшого вдосконалення і застосування у різних галузях.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Kovalenko O., Popereshnyak S., Grinenko S., Grinenko O., Radivilova T. «Methods for Assessing the Maturity Levels of Software Ecosystems». *CEUR Workshop Proceedings* Volume 2654, 2019, Pages 251-261. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=5633fba897776a6e0f3d5633fbcd3fbe **(Scopus)**.

2. Kovalenko O., Drieieva H., Simakhin V., Bondar S., Drieiev O., Zhumadilova M. «Multifractal Properties of Traffic Generator Based on Markov Chains ». *CEUR Workshop Proceedings* Volume 2588, 2019, Pages 567-579. Режим доступу: https://www.scopus.com/record/display.uri?eid=2-s2.0-85083214331&origin=AuthorNamesList&txGid=176e2cada8976a6e0f3d5633fbcd3fbe **(Scopus)**.

3. Kovalenko Oleksandr Qualitative risk analysis of software development / Oleksandr Kovalenko, Jamil Al-Azzeh, Oleksii Smirnov, Anna Kovalenko, Serhii Smirnov // Asian Journal of Information Technology. – Volume 17 Issue 3. – Medwell Journals. – 2018. – P. 218-230. ISSN: 1682-3915. URL: http://medwelljournals.com/abstract/?doi=ajit.2018.218.230 Doi: ajit.2018.218.230

4. Kovalenko Oleksandr, The mathematical model of the testing technology for DOM XSS vulnerabilities / O. Kovalenko, O. Smirnov, A.Kovalenko, S. Smirnov, V. Vialkova // Scientific & practical cyber security journal (SPCSJ) Volume 2 Issue 1, P. 22-28. Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2018 ISSN: 2587-4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/04-3-o.kovalenko-a.kovalenko-o.smirnov-s.smirnov-v.vialkova.pdf

5. Коваленко А.В. Технология тестирования DOM XSS уязвимости / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Scientific & practical cyber security journal (SPCSJ) Volume 1. Issue 1. P. 38-45 Georgia. Tbilisi. Scientific Cyber Security Association (SCSA), 2017 ISSN: 2587-

4667. URL: https://journal.scsa.ge/wp-content/uploads/2018/12/8-dom-xss-testing-technology-vulnerabilities.pdf

6. Коваленко О.В. Моделі та методи розроблення програмного забезпечення комп'ютерних систем для підвищення безпеки даних: монографія / О.В. Коваленко // К.: Вид. «КОД» – 2019. – 305 с.

7. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Информационные технологии в управлении, образовании, науке и промышленности: монография / Под редакцией профессора В.С. Пономаренко. – Х.: Видавець Рожко С.Г., 2016. – 566 с.

8. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: проблеми та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: Видавець Рожко С.Г., 2017. – 447 с.

9. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.А. Смирнов // Інформаційні технології: сучасний стан та перспективи: монографія / За загальною редакцією В.С. Пономаренка. – Х.: ТОВ «ДІСА ПЛЮС», 2018. – 461 с.

10. Коваленко А.В. Задачи распознавания ситуаций в ERP системах / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник наукових праць "Системи обробки інформації". – Випуск 4(120). – Х.: ХУПС – 2014. – С. 161-164.

11. Коваленко А.В. Методы качественного анализа и количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць "Системи обробки інформації". – Випуск 5(142). – Х.: ХУПС – 2016. – С. 153-157.

12. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць "Системи обробки інформації". –

Випуск 3(140). – Х.: ХУПС – 2016. – С. 40-42.

13. Коваленко А.В. Метод качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Наука і техніка Повітряних Сил Збройних Сил України. – Випуск 2(23). – Харків: ХУПС. – 2016. – С. 150-158.

14. Коваленко А.В. Метод количественной оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, Н.Н. Якименко, А.П. Доренский // Збірник наукових праць Харківського університету Повітряних Сил. Випуск 2 (47). – Харків: ХУПС. – 2016. – С. 128-133.

15. Коваленко А.В. Использование псевдобулевых методов бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Системи управління, навігації та зв'язку. – Випуск 1 (37). – Полтава: ПолтНТУ. – 2016. – С. 98-103.

16. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 2 (38). – Полтава: ПолтНТУ. – 2016. – С. 93-100.

17. Коваленко А.В. Технология тестирования уязвимости к SQL инъекциям / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 5 (45). – Полтава: ПолтНТУ. – 2017. – С. 66-71.

18. Коваленко А.В. Масштабирование имитационной модели технологии тестирования безопасности / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 6 (46). – Полтава: ПолтНТУ. – 2017. – С. 181-184.

19. Коваленко А.В. Имитационная модель технологии тестирования безопасности Web-приложений / А.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 1 (47). – Полтава: ПолтНТУ. – 2018. – С. 114-123.

20. Коваленко О.В. Методи якісного аналізу та кількісної оцінки ризиків розроблення програмного забезпечення/ О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 3 (49). – Полтава: ПолтНТУ. – 2018. – С. 116-125.

21. Коваленко О.В. Управління ризиками розроблення програмного

забезпечення за умови обмеженості коштів виділених на усунення помилок безпеки/ О.В. Коваленко // Техніка в сільськогосподарському виробництві, галузеве машинобудування, автоматизація. – Випуск 31. – Кропивницький: ЦНТУ. – 2018. – С. 128-140.

22. Коваленко О.В. GERT-мережевий синтез технології тестування на вразливість WEB-додатків/ О.В. Коваленко // Захист інформації. – Випуск 20(2) – К.: НАУ. – 2018. – С. 89-94.

23. Коваленко О.В. Імітаційна модель технології тестування безпеки на основі положень теорії масштабування / О.В. Коваленко // Безпека інформації. – Випуск 24 (2). – К.: НАУ. – 2018. – С. 110-117.

24. Коваленко О.В. Оцінка ефективності технології тестування безпеки / О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 2, 2018. – С. 137-141

25. Коваленко О.В. Методи та засоби управління безпекою додатків / О.В. Коваленко // Інформаційно-керуючі системи на залізничному транспорті. №4, 2018. – С. 41-44.

26. Коваленко О.В. Розробка інформаційної технології передтестової компіляції та розподілу доступу / О.В. Коваленко // Системи управління, навігації та зв'язку. – Випуск 4 (50). – Полтава: ПолтНТУ. – 2018. – С. 115-119.

27. Коваленко О.В. Аналіз та дослідження інформаційних технологій розробки програмного забезпечення/ О.В. Коваленко // Вчені записки Таврійського національного університету імені В.І. Вернадського. Серія: Технічні науки. Том 29 (68) № 5, 2018. – С. 131-137.

28. Коваленко О.В. Удосконалений метод управління ризиками розроблення програмного забезпечення на основі напівмарковської моделі прийняття рішень/ О.В. Коваленко // Сучасні інформаційні системи. – Випуск 2(3). – Харків. – 2018. – С. 41-48.

29. Коваленко О.В. Математичні моделі технології тестування DOM XSS

вразливості та вразливості до SQL ін'єкцій / О.В. Коваленко // Вісник Черкаського державного технологічного університету. Серія : Технічні науки №4, 2018. – С. 29-36.

30. Коваленко О.В. Математична модель технології тестування вразливості до SQL ін’єкцій / О.В. Коваленко // Системи управління, навігації та зв’язку. – Випуск 6 (58). – Полтава: ПолтНТУ. – 2019. – С. 43-47.

31. Коваленко О.В. Математична модель технології тестування комплексу DOM XSS вразливостей для аналітичної оцінки часових витрат / О.В. Коваленко // Центральноукраїнський науковий вісник. Технічні науки. № 2(33). с. 173-180, 2019.

32. Коваленко А.В. Проблемы анализа и оценки рисков информационной деятельности / А.В. Коваленко, А.А. Смирнов // Збірник наукових праць II міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 24-27 лютого 2016 р. – Київ: Європейський університет. – 2016. – С. 138-139.

33. Коваленко А.В. Анализ и оценка рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез «Securitea informationala 2015-2016». Conferenta internationala (editia a XII-a). Chisinau. Moldova. 3 martie 2016. – Chisinau: ADSEM. – 2016. – P. 96-102.

34. Коваленко А.В. Исследование источников и причин риска разработки программного обеспечения, этапов и работ, при выполнении которых возникает риск / А.В. Коваленко, А.А. Смирнов // Збірник тез VII всеукраїнської науково-практичної конференції "Інформатика та системні науки (ІСН-2016)". м. Полтава. 10-12 березня 2016 р. – Полтава.: ПУЕТ – 2016. – С. 264-266.

35. Коваленко А.В. Оценка показателя чистой приведенной стоимости для количественной оценки рисков проекта разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних

систем”. м. Київ. 10-11 березня 2016 р. – Київ: КНУ ім. Тараса Шевченко – 2016. – С. 81-82.

36. Коваленко А.В. Методика структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 24-25 березня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 71-72.

37. Коваленко А.В. Методы качественного анализа рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез першої міжнародної науково-практичної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2016). м. Харків. 30 березня – 1 квітня 2016 р. – Харків: НТУ «ХПІ». – 2016. – С. 6-7.

38. Коваленко А.В. Структурная идентификация рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез XVIII міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кіровоград. 15-16 квітня 2016 р. – Кіровоград: КНТУ. – 2016. – С. 175-182.

39. Коваленко А.В. Исследование разработанной методики структурной идентификации рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез VIII міжнародної науково-практичної конференції “Проблеми і перспективи розвитку IT-індустрії”. м. Харків. 28-29 квітня 2016 р. – Харків: ХНЕУ. – 2016. – С. 49.

40. Коваленко А.В. Исследование дерева рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез III міжнародної науково-практичної конференції «Інформаційна та економічна безпека» (INFECO-2016)». м. Харків. 28-30 квітня 2016 р. – Харків: ХННІ ДВНЗ «УБС». – 2016. – С. 174-178.

41. Коваленко А.В. Методы качественного анализа и количественной

оценки рисков разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Сборник тезисов XII международной конференции "Стратегия качества в промышленности и образовании". г. Варна. Болгария. 30 мая – 02 июня 2016 г – Варна. ТУВ. – 2016. – С. 585-589.

42. Коваленко А.В. Разработка метода управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Матеріали Всеукраїнської науково-практичної конференції «Кібербезпека в Україні: правові та організаційні питання». м. Одеса, 21 жовтня 2016 р. – Одеса : ОДУВС, 2016. – С.146-148.

43. Коваленко А.В. Метод управления рисками разработки программного обеспечения с использованием псевдобулевых методов бивалентного программирования / А.В. Коваленко, А.А. Смирнов // Матеріали Всеукраїнської науково-практичної конференції «Актуальні задачі та досягнення у галузі кібербезпеки». м. Кропивницький, 23-25 листопада 2016 року – Кропивницький: ЦНТУ, 2016. – С. 162.

44. Коваленко А.В. Псевдобулевые методы бивалентного программирования для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко, С.А. Смирнов // Збірник наукових праць III міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 22-25 лютого 2017 р. – Київ: Європейський університет. – 2017. – С. 158-162.

45. Коваленко А.В. Метод управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов // Збірник тез II науково-практичної конференції “Проблеми кібербезпеки інформаційно-телекомунікаційних систем”. м. Київ. 23-24 березня 2017 р. – Київ: КНУ ім. Тараса Шевченко – 2017. – С. 203-205.

46. Коваленко А.В. Алгоритмы анализа уязвимостей при управлении рисками разработки программного обеспечения / А.В. Коваленко,

А.А. Смирнов, А.С. Коваленко // Conferenta internationala (editia a XIII-a). «Securitea informationala 2017». Chisinau. Republic of Moldova. 4-5 aprilie 2017. – Chisinau: ADSEM. – 2017. – P. 19-22.

47. Коваленко А.В. Алгоритм анализа DOM XSS уязвимости при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез дев'ятнадцятого міжнародного науково-практичного семінару «Комбінаторні конфігурації та їх застосування». м. Кропивницький 7-8 квітня 2017 р. – Кропивницький: ГЛА НАУ. – 2017. – С. 125-127.

48. Коваленко А.В. Алгоритм анализа уязвимости SQL Injection для управления рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез другої міжнародної науково-технічної конференції «Проблеми науково-технічного та правового забезпечення кібербезпеки у сучасному світі» (ПНПЗК-2017). м. Харків.10-12 квітня 2017 р. – Харків: НТУ «ХПІ». – 2017. – С. 27.

49. Коваленко А.В. Метод управления рисками разработки программного обеспечения на основе алгоритмов анализа уязвимостей / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез Міжнародної науково-практичної конференції «Інформаційна безпека та комп'ютерні технології» (IS&CT). м. Кіровоград. 20-22 квітня 2017 р. Кіровоград: КНТУ. – 2017. – С. 92.

50. Коваленко А.В. Алгоритмы анализа DOM XSS уязвимости и уязвимости SQL Injection при управлении рисками разработки программного обеспечения / А.В. Коваленко, А.А. Смирнов, А.С. Коваленко // Збірник тез IX міжнародної науково-практичної конференції "Проблеми і перспективи розвитку IT-індустрії". м. Харків. 20-21 квітня 2017 р. – Харків: ХНЕУ. – 2017. – С. 61.

51. Kovalenko O.V. Method of testing the dom xss vulnerability / Kovalenko Oleksandr, Kovalenko Anna, Smirnov Oleksii, Smirnov Serhii // International

Conference «information technologies, systems and networks ITSN-2017». Chisinau, Republic of Moldova. 17 – 18 October 2017. – Chisinau: Academy of Sciences of Moldova, Military Academy of Armed Forces “Alexandru cel Bun”. – 2017. – P. 7.

52. Коваленко О.В. Метод тестування DOM XSS уразливості / О.В. Коваленко, О.А. Смірнов, А.С. Коваленко, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної інтернет-конференції «Автоматика та комп’ютерно-інтегровані технології у промисловості, телекомунікаціях, енергетиці та транспорті». м. Кропивницький. 16-17 листопада 2017 р. – Кропивницький: ЦНТУ. – 2017. – С. 198-199.

53. Коваленко О.В. GERT-модель технології тестування DOM XSS уразливості / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць IV міжнародної науково-практичної конференції «Актуальні питання забезпечення кібернетичної безпеки та захисту інформації». м. Київ. 21-24 лютого 2018 р. – Київ: Європейський університет. – 2018. – С. 65-70.

54. Коваленко О.В. Технології тестування уразливостей Web-застосунків з використанням GERT-моделі / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник тез всеукраїнської науково-практичної конференції "Комп'ютерні інтелектуальні системи та мережі (КІСМ-2018)". м. Кривий Ріг. 21-23 березня 2018 р. – Кривий Ріг.: ДВНЗ КНУ – 2018. – С. 227-230.

55. Коваленко А.В. Тестирование уязвимости Web-приложений к атаке вида межсайтовый скриптинг / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез «Securitea informationala 2018». Conferenta internationala (editia a XIV-a). Chisinau. Moldova. 20-21 martie 2018. – Chisinau: ADSEM. – 2018. – P. 54-56.

56. Коваленко А.В. Комплекс математических моделей технологии тестирования web-приложений / А.В. Коваленко, А.С. Коваленко, А.А. Смирнов, С.А. Смирнов // Збірник тез X міжнародної науково-практичної

конференції “Проблеми і перспективи розвитку ІТ-індустрії”. м. Харків. 19-20 квітня 2018 р. – Харків: ХНЕУ. – 2018. – С. 38.

57. Коваленко О.В. Розробка методу передтестової компіляції й розподілу доступу / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Збірник наукових праць III міжнародної науково-практичної конференції “Інформаційна безпека та комп’ютерні технології”, м. Кропивницькй. 19-20 квітня 2018 р. – Кропивницький: ЦНТУ. – 2018. – С. 214-215.

58. Коваленко О.В. Оцінка ефективності технологій тестування безпеки уразливостей DOM XSS й SQL-ін'єкцій / О.В. Коваленко, А.С. Коваленко, О.А. Смірнов, С.А. Смірнов // Сборник тезисов XIV международной конференции "Стратегия качества в промышленности и образовании", Варна, Болгария. 04-07 июня 2018 г – Варна. ТУВ. – 2018. – С. 271-274.

59. Коваленко О.В. Аналіз основних підходів математичного моделювання та методологій для забезпечення максимальних показників безпеки програмного забезпечення / О.В. Коваленко, А.С. Коваленко // Збірник наукових праць всеукр. наук.-практ. конф. здобувачів вищої освіти й молодих учених «Комп’ютерна інженерія і кібербезпека : досягнення та інновації, м. Кропивницькй. 27-29 листопада

Додаток А
(обов'язковий)

Технічне завдання

Зміст

1 Найменування та область застосування

Це технічне завдання розповсюджується на розробку системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

2 Підстава для розробки

Підставою для розробки служить завдання на кваліфікаційну бакалаврську роботу, видане на кафедрі кібербезпеки та програмного забезпечення (нак. № 186-02 від 28.12.2020 року).

3 Мета та призначення розробки

Метою кваліфікаційної бакалаврської роботи є розробка програмного забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів.

4 Джерела розробки

Джерелом цієї кваліфікаційної бакалаврської роботи є стосовна до теми література і існуючі аналоги.

5 Технічні вимоги

5.1 Склад продукції

Складниками розробки є:

– вибір і обґрунтування методів реалізації проекту;

– розробка програмної частин системи, а також розробка взаємодії системи з ОС та з користувачем;

– розробка програми, що реалізує спроектовані алгоритми роботи системи.

5.2 Показники призначення

Система повинна забезпечувати:

– системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів;

– цілісність даних у процесі роботи та при зберіганні;

– простий, інтуїтивно зрозумілий інтерфейс.

5.3 Вимоги до функціональних характеристик

Розроблене програмне забезпечення не повинно мати обмежень на версію драйверів та операційної системи.

5.4 Вимоги до архітектури

Компонент, що розробляється повинен використовувати системні засоби та апаратні засоби, що на даному етапі розвитку обчислювальної техніки найбільше поширені.

5.5 Вимоги до надійності

Програмні модулі написані по всім правилам, які стосуються стандартних викликів процедур, функцій, методів і форм, визначених технічною документацією на середовище розробки.

5.6 Умови експлуатації

Робочі місця користувачів ПЗ повинні задовольняти наступним умовам експлуатації:

– температура повітря: 19-20 град. по Цельсію;

– відносна вологість повітря до 80%;

– атмосферний тиск 107 кПа.

5.7 Вимоги до складу та параметрів технічних засобів

Програмне забезпечення повинно бути реалізоване на ПЕОМ архітектури IBM PC, працювати в ОС Windows XP/Vista/7/8/10 і з сумісними з цією платформою пристроями і прикладним програмним забезпечення.

5.8 Вимоги до інформаційної і програмної сумісності

Переносність програмного забезпечення повинна бути забезпечена за рахунок його реалізації стандартного інтерфейсу взаємодії з ОС, що працюють під управлінням ОС Windows XP/Vista/7/8/10.

5.8.1 Обладнання

Комп’ютер Intel® Celeron/8 Mb/1.2 Gb/SVGA 14” 1Mb або сумісні з ним.

5.8.2 Мова програмування

Середовище RAD Studio Delphi 10.4.

5.8.3 Вхідні дані

Опис алгоритму роботи запропонованої системи.

5.8.4 Вихідні дані

Робоча програма.

6 Вимоги до програмної документації

Програмна продукція повинна бути представлена у виді опису структури даних, схем та опису алгоритму, а також текстів вихідних модулів програмного забезпечення згідно ЄСПД .

7 Перелік документів, що розробляються

– Структурна схема системи – 1 аркуш.
– Функціональна схема системи – 1 аркуш.
– Діаграма процесів – 1 аркуш.
– Блок-схема алгоритму роботи програми – 2 аркуша.
– Пояснювальна записка – 67 аркушів.

8 Етапи розробки

8.1 Збір і обробка інформації по темі кваліфікаційної бакалаврської роботи. Постановка задачі на виконання кваліфікаційної бакалаврської роботи (складання ТЗ).

8.2 Проведення досліджень або експериментальних робіт для уточнення основних положень кваліфікаційної бакалаврської роботи.

8.3 Розробка функціональних схем, блок схем алгоритмів роботи програмного забезпечення.

8.4 Побудова схем взаємодії даних.

8.5 Створення прототипу ПЗ.

8.6 Віднаходження ПЗ, аналіз отриманих результатів.

8.7 Оформлення пояснювальної записки і виконання робіт по графічній частині.

11 Порядок контролю та приймання

11.1 Подання кваліфікаційної бакалаврської роботи на попередній захист 22.05.2021 р.

11.2 Подання кваліфікаційної бакалаврської роботи на захист 14.06.2021 р.

Додаток Б

(обов’язковий)

Міністерство освіти і науки України

Центральноукраїнський національний технічний університет

ЗАТВЕРДЖУЮ

Керівник кваліфікаційної бакалаврської роботи

_____ Смірнов О.А.

Програмне забезпечення системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів

Лістинг програми

Код документу 12

Носій: CD/DVD-диск

Загальна кількість аркушів: 63

Літера: РП

Кропивницький – 2021 року

NNTA_Editor.pas – розробка системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів

```
unit NNTA_Editor;

interface

uses
  Classes,
  DesignIntf, DesignEditors,
  NNTA_Comp, NNTA_EditorForm,
  SysUtils, Dialogs, Controls, NNTA_EditorFieldsForm;

type
  TNeuronsInLayerFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

  TFileNameFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    procedure Edit; override;
  end;

  TOptionsFieldsProperty = class(TStringProperty)
  public
    function GetAttributes : TPropertyAttributes; override;
    function GetValue : string; override;
    procedure Edit; override;
  end;

procedure Register;

implementation

function TOptionsFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

procedure TOptionsFieldsProperty.Edit;
var
  i: integer;
  xNeuralNetExtended: TNeuralNetExtended;
  frmNeuroFields: TfrmNeuroFields;
  xCurrent: integer;
begin
  frmNeuroFields := TfrmNeuroFields.Create(nil);
  try
    with frmNeuroFields do
    begin
      xNeuralNetExtended := (GetComponent(0) as TNeuralNetExtended);
      for i := 0 to xNeuralNetExtended.AvailableFieldsCount - 1 do
        ltbFieldName.Items.Add(xNeuralNetExtended.Fields[i].Name);
      xCurrent := 0;
      rdgFieldType.ItemIndex := xNeuralNetExtended.Fields[xCurrent].Kind;
      rdgNormType.ItemIndex := xNeuralNetExtended.Fields[xCurrent].NormType;
      edtAlpha.Text := FloatToStr(xNeuralNetExtended.Fields[xCurrent].Alpha);
      sttMin.Caption :=
FloatToStr(xNeuralNetExtended.Fields[xCurrent].ValueMin);
      sttMax.Caption :=
FloatToStr(xNeuralNetExtended.Fields[xCurrent].ValueMax);
```

```
      NeuralNetExtended := xNeuralNetExtended;
      ShowModal;
    end;
  except
    frmNeuroFields.Free;
  end;
end;

function TOptionsFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TNeuralNetExtended) do
    if AvailableFieldsCount > 1 then
    begin
      for i := 0 to AvailableFieldsCount - 1 do
        Result := Result + Fields[i].Name + ',';
      Result[Length(Result)] := ']'
    end
    else
      Result[Length(Result) + 1] := ']';
end;

function TNeuronsInLayerFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

function TNeuronsInLayerFieldsProperty.GetValue;
var
  i: integer;
begin
  // внести зміни
  Result := '[';
  with (GetComponent(0) as TNeuralNetBP) do
    if LayerCount > 0 then
    begin
      for i := 0 to LayerCount - 1 do
        Result := Result + IntToStr(LayersBP[i].NeuronCount) + ',';
      Result[Length(Result)] := ']';
    end
    else
      Result[Length(Result) + 1] := ']';
end;

procedure TNeuronsInLayerFieldsProperty.Edit;
var
  i: integer;
  xPreviousCount: integer;
  xNeuralNetBP: TNeuralNetBP;
  frmNeuronsInLayer: TfrmNeuronsInLayer;
  xChangesMade: boolean;
begin
  xChangesMade := False;
  frmNeuronsInLayer := TfrmNeuronsInLayer.Create(nil);
  try
    with frmNeuronsInLayer do
    begin
      xNeuralNetBP := (GetComponent(0) as TNeuralNetBP);
      speLayers.Value := xNeuralNetBP.LayerCount;
      stgNeuronsInLayer.RowCount := xNeuralNetBP.LayerCount + 1;
      for i := 0 to xNeuralNetBP.LayerCount - 1 do
      begin
        stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
        stgNeuronsInLayer.Cells[1, i + 1] :=
IntToStr(xNeuralNetBP.LayersBP[i].NeuronCount);
      end;
```

```
      if ShowModal = mrOk then
      begin
        xNeuralNetBP.ResetLayers;
        if speLayers.Value = 0 then
        begin
          xNeuralNetBP.ResetLayers;
          Exit;
        end;
        if xNeuralNetBP.LayerCount <> speLayers.Value then
          xChangesMade := True
        else
          for i := 0 to xNeuralNetBP.LayerCount - 1 do
            if xNeuralNetBP.LayersBP[i].NeuronCount <>
StrToInt(stgNeuronsInLayer.Cells[1, i + 1]) then
            begin
              xChangesMade := True;
              Break;
            end;
        if xChangesMade then
        begin
          if xNeuralNetBP.LayerCount = speLayers.Value then
            for i := 0 to speLayers.Value - 1 do
              xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i +
1]
          else
            if xNeuralNetBP.LayerCount < speLayers.Value then
            begin
              for i := 0 to xNeuralNetBP.LayerCount - 1 do
                xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i +
1];
              for i := xNeuralNetBP.LayerCount to speLayers.Value - 1 do
                xNeuralNetBP.AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i +
1]));
            end
            else
            begin
              if speLayers.Value > 0 then
                for i := 0 to speLayers.Value - 1 do
                  xNeuralNetBP.NeuronsInLayer[i] := stgNeuronsInLayer.Cells[1, i
+ 1];
              xPreviousCount := xNeuralNetBP.LayerCount -  speLayers.Value;
              for i := 1 to xPreviousCount do
                xNeuralNetBP.DeleteLayer(xNeuralNetBP.LayerCount - 1);
            end;
          if not xNeuralNetBP.AutoInit then
            xNeuralNetBP.AutoInit := True;
        end;
      end;
    end;
  except
    frmNeuronsInLayer.Free;
  end;
end;

function TFileNameFieldsProperty.GetAttributes;
begin
  Result := [paDialog];
end;

procedure TFileNameFieldsProperty.Edit;
var
  xOpenDialog: TOpenDialog;
begin
  xOpenDialog := TOpenDialog.Create(nil);
  if UpperCase(GetName) = 'FILENAME' then
    xOpenDialog.Filter := 'Нейронна мережа(*.nnw)|*.nnw|всі файли (*.*)|*.*';
  if UpperCase(GetName) = 'SOURCEFILENAME' then
    xOpenDialog.Filter := 'Текстові файли (*.txt)|*.txt|всі файли (*.*)|*.*';
```

```
  if xOpenDialog.Execute then
  begin
    if UpperCase(GetName) = 'FILENAME' then
      (GetComponent(0) as TNeuralNetExtended).FileName := xOpenDialog.FileName;
    if UpperCase(GetName) = 'SOURCEFILENAME' then
      (GetComponent(0) as TNeuralNetExtended).SourceFileName :=
xOpenDialog.FileName;
  end;
  xOpenDialog.Free;
end;

procedure Register;
begin
  RegisterPropertyEditor(TypeInfo(TStrings), TNeuralNetBP, 'NeuronsInLayer',
TNeuronsInLayerFieldsProperty);
  RegisterPropertyEditor(TypeInfo(TFileName), TNeuralNetExtended, '',
TFileNameFieldsProperty);
  RegisterPropertyEditor(TypeInfo(string), TNeuralNetExtended, 'Options',
TOptionsFieldsProperty);
end;

end.
```

NNTA_EditorForm.pas – редактор системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів

```
unit NNTA_EditorForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, StdCtrls, ExtCtrls, NNTA_Comp, Spin, NNTA_Types;

type
  TfrmNeuronsInLayer = class(TForm)
    Bevel1: TBevel;
    btnOk: TButton;
    btnCancel: TButton;
    stgNeuronsInLayer: TStringGrid;
    Label1: TLabel;
    speLayers: TSpinEdit;
    procedure stgNeuronsInLayerGetEditMask(Sender: TObject; ACol,
      ARow: Integer; var Value: String);
    procedure stgNeuronsInLayerSetEditText(Sender: TObject; ACol,
      ARow: Integer; const Value: String);
    procedure speLayersChange(Sender: TObject);
    procedure FormCreate(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  frmNeuronsInLayer: TfrmNeuronsInLayer;

implementation

{$R *.DFM}

procedure TfrmNeuronsInLayer.stgNeuronsInLayerGetEditMask(Sender: TObject;
  ACol, ARow: Integer; var Value: String);
begin
  Value := '0000';
end;

procedure TfrmNeuronsInLayer.stgNeuronsInLayerSetEditText(Sender: TObject;
  ACol, ARow: Integer; const Value: String);
begin
{  with stgNeuronsInLayer do
    try
      Cells[ACol, ARow] := IntToStr(StrToInt(trim(Value)));
    except
      Cells[ACol, ARow] := IntToStr(DefaultNeuronCount);
    end;}
end;

procedure TfrmNeuronsInLayer.speLayersChange(Sender: TObject);
var
  i: integer;
begin
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
  for i := 1 to stgNeuronsInLayer.RowCount do
    if trim(stgNeuronsInLayer.Cells[1, i]) = '' then
    begin
      stgNeuronsInLayer.Cells[0, i] := IntToStr(i);
      stgNeuronsInLayer.Cells[1, i] := IntToStr(DefaultNeuronCount);
    end;
end;
```

```
procedure TfrmNeuronsInLayer.FormCreate(Sender: TObject);
begin
  stgNeuronsInLayer.Cells[0,0] := '# шару';
  stgNeuronsInLayer.Cells[1,0] := 'нейронів';
end;

end.
```

NNTA_EditorFieldsForm.pas – редактор властивостей полів нейроної мережі

```
unit NNTA_EditorFieldsForm;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  StdCtrls, ExtCtrls, NNTA_Comp;

type
  TfrmNeuroFields = class(TForm)
    ltbFieldName: TListBox;
    rdgFieldType: TRadioGroup;
    rdgNormType: TRadioGroup;
    Bevel1: TBevel;
    Label1: TLabel;
    Label2: TLabel;
    Label3: TLabel;
    sttMin: TStaticText;
    sttMax: TStaticText;
    edtAlpha: TEdit;
    btnOk: TButton;
    btnCancel: TButton;
    Bevel2: TBevel;
    Label4: TLabel;
    procedure ltbFieldNameClick(Sender: TObject);
    procedure rdgFieldTypeClick(Sender: TObject);
    procedure rdgNormTypeClick(Sender: TObject);
    procedure edtAlphaChange(Sender: TObject);
  private
    { Private declarations }
  public
    NeuralNetExtended: TNeuralNetExtended;
    { Public declarations }
  end;

var
  frmNeuroFields: TfrmNeuroFields;

implementation

{$R *.DFM}

procedure TfrmNeuroFields.ltbFieldNameClick(Sender: TObject);
begin
  rdgFieldType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind;
  rdgNormType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType;
  edtAlpha.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha);
  sttMin.Caption :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMin);
  sttMax.Caption :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMax);
end;

procedure TfrmNeuroFields.rdgFieldTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
  else
```

```
    if ltbFieldName.SelCount > 1 then
    for i := 0 to ltbFieldName.Items.Count - 1 do
      if ltbFieldName.Selected[i] then
        NeuralNetExtended.Fields[i].Kind := rdgFieldType.ItemIndex;
end;

procedure TfrmNeuroFields.rdgNormTypeClick(Sender: TObject);
var
  i: integer;
begin
  if ltbFieldName.SelCount = 1 then
    NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
  else
    if ltbFieldName.SelCount > 1 then
    for i := 0 to ltbFieldName.Items.Count - 1 do
      if ltbFieldName.Selected[i] then
        NeuralNetExtended.Fields[i].NormType := rdgNormType.ItemIndex;
end;

procedure TfrmNeuroFields.edtAlphaChange(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha :=
StrToFloat(edtAlpha.Text);
end;

end.
```

NNTA_Types.pas – ініціалізація базових констант

```
unit NNTA_Types;

interface

const

  DefaultAlpha = 1;               // Параметр активаційної функції
  DefaultEpochCount = 10000;      // Кількість етапів для навчання
  DefaultErrorValue = 0.05;       // Помилка за замовчуванням для всіх видів
  DefaultHopfLayerCount = 2;      // Кількість шарів у мережі Хопфилда
  DefaultLayerCount = 0;          // Мінімальна кількість шарів у мережі back-
propagation
  DefaultMaxIterCount = 10;       // Максимальна кількість ітерацій в алгоритмі
Хопфилда
  DefaultMomentum = 0.9;          // Імпульс - момент
  DefaultNeuronCount = 0;         // Кількість нейронів у прихованому шарі за
замовчуванням
  DefaultPatternCount = 0;        // Кількість прикладів
  DefaultTeachRate = 0.1;         // Швидкість навчання
  DefaultTeachIdentCount = 100;   // Необхідний відсоток розпізнаних прикладів з
навчальної множини
  DefaultTestIdentCount = 100;    // Необхідний відсоток розпізнаних прикладів з
тестової множини
  DefaultUseForTeach = 100;       // Відсоток прикладів використовуваних як
навчальна множина
  DefaultDeltaBarAcceleratingConst = 0.095;
  DefaultDeltaBarDecFactor = 0.85;
  DefaultRPropInitValue = 0.1;
  DefaultRPropMaxStepSize = 50;
  DefaultRPropMinStepSize = 1 E-6;
  DefaultRPropDecFactor = 0.5;
  DefaultRPropIncFactor = 1.2;
  DefaultSuperSABDecFactor = 0.5;
  DefaultSuperSABIncFactor = 1.05;

  SensorLayer = 0;                // Сенсорний шар
  BiasNeuron = 1;                 // Зсув у мережі back-propagation

  Separators = ['.', ','];
  Letters    = ['a'..'z'];
  Capitals   = ['A'..'Z'];
  DigitChars = ['0'..'9'];
  SpaceChar = #9;

resourcestring

  SFieldNorm = 'Не існує типу нормалізації %d';
  SFieldKind = 'Не існує типу поля %d';
  SFieldIndexRange = 'Неправильно зазначений номер поля %d';
  SInFieldCount = 'Неправильно встановлена кількість вхідних полів';
  SInNeuronCount = 'Неправильно встановлена кількість вхідних нейронів';
  SInVectorCount = 'Неправильно встановлена розмірність вхідного вектора';
  SLayerRangeIndex = 'Неправильно зазначений номер шару %d';
  SNeuronRangeIndex = 'Неправильно зазначений номер нейрона %d';
  SNeuronCount = 'Неправильно зазначена кількість нейронів';
  SOutFieldCount = 'Неправильно встановлена кількість вихідних полів';
  SOutNeuronCount = 'Неправильно встановлена кількість вихідних нейронів';
  SOutVectorCount = 'Неправильно встановлена розмірність вихідного вектора';
  SPatternRangeIndex = 'Вихід за межі масиву прикладів %d';
  SStreamCannotRead = 'Помилка читання з потоку';
  SWeightRangeIndex = 'Неправильно зазначений номер ваги %d';
  SWrongFileName = 'Неправильно зазначене ім'я файлу %s';
  SCannotBeNumber = 'Помилка, вираз %s неможливо привести до числового типу';
  SBPStopCondition = 'Не задана умова зупинки процесу навчання';
```

```
type

  TVectorInt = array of integer;
  TVectorFloat = array of double;
  TVectorString = array of string;
  TMatrixInt = array of array of integer;
  TMatrixFloat = array of array of double;
  TNormalize = (nrmLinear, nrmSigmoid, nrmAuto,  nrmNone,
                nrmLinearOut, nrmAutoOut);
  TNeuroFieldType = (fdInput, fdOutput, fdNone);

implementation


end.
```

NNTA_Comp.pas – формування бібліотеки шаблонів та дослідження системи пошуку інформації за допомогою технології нейронних мереж з архітектурою трансформерів

```
unit NNTA_Comp;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, PumpData, NNTA_Types, IniFiles, Math;

type

  // Класи виключень
  EInOutDimensionError = class(Exception);
  ENeuronCountError  = class(Exception);
  ENeuronNotEqualFieldError  = class(Exception);
  EBPStopCondition  = class(Exception);

  // Процедурні типи
  TActivation = function (Value: double): double of object;

  // Упереджуюче оголошення класів
  TNeuron = class;
  TLayer = class;

  // Базовий клас нейрона

  TNeuron = class(TObject)
  private
    FOutput: double;
    // Вектор ваг
    FWeights: TVectorFloat;
    // Покажчик на шар, у якому перебуває нейрон
    Layer: TLayer;
    function GetWeights(Index: integer): double;
    procedure SetWeights(Index: integer; Value: double);
    procedure SetWeightCount(const Value: integer);
  public
    constructor Create(ALayer: TLayer); virtual;
    destructor Destroy; override;
    // Ініціалізація ваг
    procedure InitWeights;  virtual;
    // Зважена сума
    procedure ComputeOut(const AInputs: TVectorFloat); virtual;
    property Output: double read FOutput write FOutput;
    property WeightCount: integer  write SetWeightCount;
    property Weights[Index: integer]: double read GetWeights write SetWeights;
  end;

  // Клас нейрона для мережі Хопфілда

  TNeuronHopf = class(TNeuron)
  public
    procedure ComputeOut(const AInputs: TVectorFloat); override;
  end;

  // Клас нейрона для мережі

  TNeuronBP = class(TNeuron)
  private
    // Локальна помилка
    FDelta: double;
    // Значення швидкості навчання на попередньому етапі
    FLearningRate: TVectorFloat;
    // Значення частинної похідної на попередньому етапі
    FPrevDerivative: TVectorFloat;
```

```
    // Значення корекції ваги на попередньому етапі
    FPrevUpdate: TVectorFloat;
    // Функція активації
    FOnActivation: TActivation;
    // Похідна функції активації
    FOnActivation: TActivation;
    function GetPrevUpdate(Index: integer): double;
    function GetPrevDerivative(Index: integer): double;
    function GetLearningRate(Index: integer): double;
    function GetPrevUpdateCount: integer;
    procedure SetPrevDerivative(Index: integer; const Value: double);
    procedure SetPrevDerivativeCount(const Value: integer);
    procedure SetDelta(Value: double);
    procedure SetPrevUpdate(Index: integer; Value: double);
    procedure SetPrevUpdateCount(const Value: integer);
    procedure SetLearningRate(Index: integer; const Value: double);
    procedure SetLearningRateCount(const Value: integer);
  public
    destructor Destroy; override;
    procedure ComputeOut(const AInputs: TVectorFloat); override;
    property Delta: double read FDelta write SetDelta;
    property LearningRate[Index: integer]: double read GetLearningRate write
SetLearningRate; //
    property LearningRateCount: integer write SetLearningRateCount;
    property PrevDerivativeCount: integer write SetPrevDerivativeCount;
    property PrevDerivative[Index: integer]: double read GetPrevDerivative write
SetPrevDerivative; //
    property PrevUpdateCount: integer read GetPrevUpdateCount write
SetPrevUpdateCount;
    property PrevUpdate[Index: integer]: double read GetPrevUpdate write
SetPrevUpdate;
    property OnActivation: TActivation read FOnActivation write FOnActivation;
    property OnActivation: TActivation read FOnActivation write FOnActivation;
  end;

  // Базовий клас шару
  TLayer = class(TPersistent)
  private
    FNumber: integer;
    // Розмірність NeuronCount
    FNeurons: array of TNeuron;
    function GetNeurons(Index: integer): TNeuron;
    function GetNeuronCount: integer;
    procedure SetNeurons(Index: integer; Value: TNeuron);
    procedure SetNeuronCount(Value: integer);
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); virtual;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
    property Neurons[Index: integer]: TNeuron read GetNeurons write SetNeurons;
    property NeuronCount: integer read GetNeuronCount write SetNeuronCount;
  end;

  // Клас шару для мережі Хопфілда
  TLayerHopf = class(TLayer)
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); override;
  end;

  // Клас шару для мережі
  TLayerBP = class(TLayer)
  private
    function GetNeuronsBP(Index: integer): TNeuronBP;
    procedure SetNeuronsBP(Index: integer; Value: TNeuronBP);
  public
    constructor Create(ALayerNumber: integer; ANeuronCount: integer); override;
    destructor Destroy; override;
    procedure Assign(Source: TPersistent); override;
```

```
    property NeuronsBP[Index: integer]: TNeuronBP read GetNeuronsBP write
SetNeuronsBP;
  end;

  // Базовий клас мережі
  TNeuralNet = class(TComponent)
  private
    // Масив шарів
    FLayers: array of TLayer;
    // Число вибірок
    FPatternCount: integer;
    // Розмірність FPatternCount, InputNeuronCount
    FPatternsInput: TMatrixFloat;
    // Розмірність FPatternCount, OutputNeuronCount
    FPatternsOutput: TMatrixFloat;
    function GetLayers(Index: integer): TLayer;
    function GetOutputNeuronCount: integer;
    function GetPatternsOutput(PatternIndex: integer; OutputIndex: integer):
double;
    function GetPatternsInput(PatternIndex: integer; InputIndex: integer):
double;
    procedure SetLayers(Index: integer; Value: TLayer);
    procedure SetPatternsInput(PatternIndex: integer; InputIndex: integer;
Value: double);
    procedure SetPatternsOutput(PatternIndex: integer; InputIndex: integer;
Value: double);
  protected
    function GetLayerCount: integer; virtual;
    function GetInputNeuronCount: integer; virtual;
    procedure Clear; virtual;
    procedure ResizeInputDim; virtual;
    procedure ResizeOutputDim; virtual;
    procedure SetPatternCount(const Value: integer); virtual;
    procedure SetLayerCount(Value: integer); virtual;
    property PatternCount: integer read FPatternCount write SetPatternCount;
  public
    destructor Destroy; override;
    procedure AddLayer(ANeurons: integer); virtual; abstract;
    procedure AddPattern(const AInputs: TVectorFloat; const AOutputs:
TVectorFloat); overload; virtual;
    procedure DeleteLayer(Index: integer); virtual; abstract;
    procedure DeletePattern(Index: integer); virtual;
    procedure Init(const ANeuronsInLayer: TVectorInt); overload; virtual;
    property InputNeuronCount: integer read GetInputNeuronCount;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Layers[Index: integer]: TLayer read GetLayers write SetLayers;
    property OutputNeuronCount: integer read GetOutputNeuronCount;
    property PatternsInput[PatternIndex: integer; InputIndex: integer]: double
read GetPatternsInput write SetPatternsInput;
    property PatternsOutput[PatternIndex: integer; InputIndex: integer]: double
read GetPatternsOutput write SetPatternsOutput;
    procedure ResetPatterns; virtual;
  end;

  // Клас мережі Хопфілда
  TNeuralNetHopf = class(TNeuralNet)
  private
    FAutoInit: boolean;
    FInputNeuronCount: integer;
    FMaxIterCount: integer;
    FPatternCount: integer;
    FPatterns: TMatrixInt;
    FOnAfterInit: TNotifyEvent;
    FOnBeforeInit: TNotifyEvent;
    FOnPatternRecognized: TNotifyEvent;
    function GetInput(Index: integer): double;
    function GetPatterns(InputIndex: integer; PatternIndex: integer): integer;
    function Stabled: boolean;
    procedure SetInput(Index: integer; Value: double);
```

```
    procedure SetPatterns(InputIndex: integer; PatternIndex: integer; Value:
integer);
  protected
    function GetInputNeuronCount: integer; override;
    function GetLayerCount: integer; override;
    procedure SetInputNeuronCount(Value: integer);
    procedure SetPatternCount(const Value: integer); override;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure AddPattern(const ANewPattern: TVectorInt); reintroduce; overload;
    procedure Calc; virtual;
    procedure DeletePattern(Index: integer); override;
    procedure Init; reintroduce; overload;
    procedure InitWeights; virtual;
    procedure ResetPatterns; override;
    procedure ResizePatternsDim; virtual;
    property Input[Index: integer]: double read GetInput write SetInput;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Patterns[InputIndex: integer; PatternIndex: integer]: integer read
GetPatterns write SetPatterns;
  published
    property AutoInit: boolean read FAutoInit write FAutoInit;
    property InputNeuronCount: integer read GetInputNeuronCount write
SetInputNeuronCount;
    property MaxIterCount: integer read FMaxIterCount write FMaxIterCount;
    property OnAfterInit: TNotifyEvent read FOnAfterInit write FOnAfterInit;
    property OnBeforeInit: TNotifyEvent read FOnBeforeInit write FOnBeforeInit;
    property OnPatternRecognized: TNotifyEvent read FOnPatternRecognized write
FOnPatternRecognized;
    property PatternCount: integer read FPatternCount write SetPatternCount;
  end;

  // Клас мережі
  TNeuralNetBP = class(TNeuralNet)
  private
    // Коефіцієнт крутості граничної сигмоїдальної функції
    FAlpha: double;
    // Прапор автоініціалізації топології мережі
    FAutoInit: boolean;
    // Прапор продовження навчання
    FContinueTeach: boolean;
    // Бажаний вихід нейромережі розмірність OutputNeuronCount
    FDesiredOut: TVectorFloat;
    // Прапор зупинки при досягненні FEpochCount
    FEpoch: boolean;
    // Лічильник етапів (пред'явлення мережі всіх прикладів з навчальної
вибірки)
    FEpochCount: integer;
    // Номер поточної епохи
    FEpochCurrent: integer;
    // Значення помилки, при якій приклад вважається розпізнаним
    FIdentError: double;
    // Значення максимальної помилки на навчальній множині
    FMaxTeachResidual: double;
    // Значення максимальної помилки на тестовій множині
    FMaxTestResidual: double;
    // Значення середньої помилки на навчальній множині
    FMidTeachResidual: double;
    // Значення середньої помилки на тестовій множині
    FMidTestResidual: double;
    // Помилка на навчальній множині
    FTeachError: double;
    // Коефіцієнт інерційності
    FMomentum: double;
    // Кількість нейронів у шарах
    FNeuronsInLayer: TStrings;
    // Подія після ініціалізації
    FOnAfterInit: TNotifyEvent;
```

```
    FOnAfterNeuronCreated: TNotifyEvent;
    // Подія після навчання
    FOnAfterTeach: TNotifyEvent;
    // Подія до ініціалізації
    FOnBeforeInit: TNotifyEvent;
    // Подія до початку навчання
    FOnBeforeTeach: TNotifyEvent;
    // Подія після проходження одного етапу
    FOnEpochPassed: TNotifyEvent;
    // Число прикладів у навчальній безлічі
    FPatternCount: integer;
    // Масив утримуючий псевдовипадкову послідовність
    FRandomOrder: TVectorInt;
    // Лічильник розпізнаних прикладів на навчальній множині
    FRecognizedTeachCount: integer;
    // Лічильник розпізнаних прикладів на навчальній множині
    FRecognizedTestCount: integer;
    // Прапор зупинки навчання
    FStopTeach: boolean;
    FTeachStopped: boolean;
    // Коефіцієнт швидкості навчання - величина градієнтного кроку
    FTeachRate: double;
    // Число прикладів у тестовій множині
    FTestSetPatternCount: integer;
    // Розмірність FTestSetPatternCount, InputNeuronCount
    FTestSetPatterns: TMatrixFloat;
    // Розмірність FTestSetPatternCount, InputNeuronCount
    FTestSetPatternsOut: TMatrixFloat;
    function GetDesiredOut(Index: integer): double;
    function GetLayersBP(Index: integer): TLayerBP;
    function GetTestSetPatterns(InputIndex, PatternIndex: integer): double;
    function GetTestSetPatternsOut(InputIndex, PatternIndex: integer): double;
    procedure NeuronCountError;
    procedure NeuronsInLayerChange(Sender: TObject);
    procedure SetAlpha(Value: double);
    procedure SetDesiredOut(Index: integer; Value: double);
    procedure SetEpochCount(Value: integer);
    procedure SetLayersBP(Index: integer; Value: TLayerBP);
    procedure SetMomentum(Value: double);
    procedure SetTeachRate(Value: double);
    procedure SetTestSetPatternCount(const Value: integer);
    procedure SetTestSetPatterns(InputIndex, PatternIndex: integer; const Value:
double);
    procedure SetTestSetPatternsOut(InputIndex, PatternIndex: integer; const
Value: double);
    // Перетасування набору даних
    procedure Shuffle;
  protected
    function GetLayerCount: integer; override;
    function GetOutput(Index: integer): double; virtual;
    // Активаційна функція
    function Activation(Value: double): double; virtual;
    // Похідна активаційної функції
    function Activation(Value: double): double; virtual;
    // Середня квадратична помилка
    function QuadError: double; virtual;
    // Підстроювання ваг
    procedure AdjustWeights; virtual;
    // Розраховує  локальну помилку - дельту
    procedure CalcLocalError; virtual;
    // Перевірка мережі на тестовій множині
    procedure CheckTestSet; virtual;
    procedure DoOnAfterInit; virtual;
    procedure DoOnAfterNeuronCreated(ALayerIndex, ANeuronIndex: integer);
virtual;
    procedure DoOnAfterTeach; virtual;
    procedure DoOnBeforeInit; virtual;
    procedure DoOnBeforeTeach; virtual;
    procedure DoOnEpochPassed; virtual;
```

```
    // Ініціалізація ваг мережі псевдовипадковими значеннями
    procedure InitWeights; virtual;
    // Пред'явлення мережі вхідних значень приклада
    procedure LoadPatternsInput(APatternIndex :integer); virtual;
    // Пред'явлення мережі вхідних значень приклада
    procedure LoadPatternsOutput(APatternIndex :integer); virtual;
    // Поширює сигнал у прямому напрямку
    procedure Propagate; virtual;
    // Установка значень за замовчуванням
    procedure SetDefaultProperties; virtual;
    procedure SetPatternCount(const Value: integer); override;
    // Струс мережі
    procedure ShakeUp; virtual;
    property TeachStopped: boolean read FTeachStopped write FTeachStopped;
  public
    constructor Create(AOwner: TComponent); override;
    destructor Destroy; override;
    procedure AddLayer(ANeurons: integer); override;
    procedure Compute(AVector: TVectorFloat); virtual;
    procedure DeleteLayer(Index: integer); override;
    procedure Init; reintroduce; overload;
    procedure ResetLayers; virtual;
    procedure TeachOffLine; virtual;
    property DesiredOut[Index: integer]: double read GetDesiredOut write
SetDesiredOut;
    property EpochCurrent: integer read FEpochCurrent;
    property IdentError: double read FIdentError write FIdentError;
    property LayersBP[Index: integer]: TLayerBP read GetLayersBP write
SetLayersBP;
    property LayerCount: integer read GetLayerCount write SetLayerCount;
    property Output[Index: integer]: double read GetOutput;
    property StopTeach: boolean read FStopTeach write FStopTeach;
    property TeachError: double read FTeachError;
    property MaxTeachResidual: double read FMaxTeachResidual;
    property MaxTestResidual: double read FMaxTestResidual;
    property MidTeachResidual: double read FMidTeachResidual;
    property MidTestResidual: double read FMidTestResidual;
    property RecognizedTeachCount: integer read FRecognizedTeachCount;
    property RecognizedTestCount: integer read FRecognizedTestCount;
    property TestSetPatternCount: integer read FTestSetPatternCount write
SetTestSetPatternCount;
    property TestSetPatterns[InputIndex: integer; PatternIndex: integer]: double
read GetTestSetPatterns write SetTestSetPatterns;
    property TestSetPatternsOut[InputIndex: integer; PatternIndex: integer]:
double read GetTestSetPatternsOut write SetTestSetPatternsOut;
  published
    property Alpha: double read FAlpha write SetAlpha;
    property AutoInit: boolean read FAutoInit write FAutoInit;
    property ContinueTeach: boolean read FContinueTeach write FContinueTeach;
    property Epoch: boolean read FEpoch write FEpoch;
    property EpochCount: integer read FEpochCount write SetEpochCount;
    property Momentum: double read FMomentum write SetMomentum;
    property NeuronsInLayer: TStrings read FNeuronsInLayer write
FNeuronsInLayer;
    property OnAfterInit: TNotifyEvent read FOnAfterInit write FOnAfterInit;
    property OnAfterNeuronCreated: TNotifyEvent read FOnAfterNeuronCreated write
FOnAfterNeuronCreated;
    property OnAfterTeach: TNotifyEvent read FOnAfterTeach write FOnAfterTeach;
    property OnBeforeInit: TNotifyEvent read FOnBeforeInit write FOnBeforeInit;
    property OnBeforeTeach: TNotifyEvent read FOnBeforeTeach write
FOnBeforeTeach;
    property OnEpochPassed: TNotifyEvent read FOnEpochPassed write
FOnEpochPassed;
    property PatternCount: integer read FPatternCount write SetPatternCount;
    property TeachRate: double read FTeachRate write SetTeachRate;
  end;

  // Клас мережі back-propagation TNeuralNetExtended }
  TNeuralNetExtended = class(TNeuralNetBP)
```

```
private
  // Файл даних
  FNeuroDataSource: TNeuroDataSource;
  // Ім'я файлу даних *.txt
  FSourceFileName: TFileName;
  // Ім'я конфігураційного файлу *.nnw
  FFileName: TFileName;
  // Конфігураційний файл
  FNnwFile: TIniFile;
  // Поля
  FFields: TNeuroFields;
  // Кількість доступних полів
  FAvailableFieldsCount: integer;
  FMaxTeachError: boolean;
  FMaxTeachErrorValue: double;
  FMaxTestError: boolean;
  FMaxTestErrorValue: double;
  FMidTeachError: boolean;
  FMidTeachErrorValue: double;
  FMidTestError: boolean;
  FMidTestErrorValue: double;
  FOptions: string;
  FSettingsLoaded: boolean;
  FTestAsValid: boolean;
  FTeachIdent: boolean;
  FTeachIdentCount: integer;
  FTestIdent: boolean;
  FTestIdentCount: integer;
  FUseForTeach: integer;
  FIdentError: double;
  FRealOutputIndex: TVectorInt;
  FRealInputIndex: TVectorInt;
  function GetFields(Index: integer): TNeuroField;
  function GetInputFieldCount: integer;
  function GetOutputFieldCount: integer;
  function GetRealInputIndex(Index: integer): integer;
  function GetRealOutputIndex(Index: integer): integer;
  procedure SetFields(Index: integer; Value: TNeuroField);
  procedure SetFileName(Value: TFilename);
  procedure SetAvailableFieldsCount(Value: integer);
  procedure SetUseForTeach(const Value: integer);
  procedure SetTeachIdentCount(const Value: integer);
  procedure SetRealOutputIndex(Index: integer; const Value: integer);
  procedure SetRealOutputIndexCount(const Value: integer);
  procedure SetRealInputIndex(Index: integer; const Value: integer);
  procedure SetRealInputIndexCount(const Value: integer);
protected
  function GetOutput(Index: integer): double; override;
  procedure DoOnBeforeTeach; override;
  procedure DoOnEpochPassed; override;
  procedure SetDefaultProperties; override;
public
  constructor Create(AOwner: TComponent); override;
  destructor Destroy; override;
  procedure ComputeUnPrepData(AVector: TVectorFloat);
  // Завантажує дані з текстового файлу
  procedure LoadDataFrom;
  // Завантажує настроювання мережі
  procedure LoadNetwork;
  // Завантажує настроювання мережі
  procedure LoadPhase1;
  // Завантажує настроювання мережі
  procedure LoadPhase2;
  // Завантажує настроювання мережі
  procedure LoadPhase4;
  // Нормалізує набір даних
  procedure NormalizeData;
  // Зберігає настроювання мережі
  procedure SaveNetwork;
```

```
    // Зберігає настроювання мережі
    procedure SavePhase1;
    // Зберігає настроювання мережі
    procedure SavePhase2;
    // Зберігає настроювання мережі
    procedure SavePhase4;
    // Навчання нейронної мережі
    procedure Train;
    property AvailableFieldsCount: integer read FAvailableFieldsCount write
SetAvailableFieldsCount;
    property Fields[Index: integer]: TNeuroField read GetFields write SetFields;
    property InputFieldCount: integer read GetInputFieldCount;
    property OutputFieldCount: integer read GetOutputFieldCount;
    property SettingsLoaded: boolean read FSettingsLoaded write FSettingsLoaded;
    property RealOutputIndex[Index: integer]: integer read GetRealOutputIndex
write SetRealOutputIndex;
    property RealOutputIndexCount: integer write SetRealOutputIndexCount;
    property RealInputIndex[Index: integer]: integer read GetRealInputIndex
write SetRealInputIndex;
    property RealInputIndexCount: integer write SetRealInputIndexCount;
    property NnwFile: TIniFile read FNnwFile write FNnwFile;
  published
    property FileName: TFileName read FFileName write SetFileName;
    property IdentError: double read FIdentError write FIdentError;
    property MaxTeachError: boolean read FMaxTeachError write FMaxTeachError;
    property MaxTeachErrorValue: double read FMaxTeachErrorValue write
FMaxTeachErrorValue;
    property MaxTestError: boolean read FMaxTestError write FMaxTestError;
    property MaxTestErrorValue: double read FMaxTestErrorValue write
FMaxTestErrorValue;
    property MidTeachError: boolean read FMidTeachError write FMidTeachError;
    property MidTeachErrorValue: double read FMidTeachErrorValue write
FMidTeachErrorValue;
    property MidTestError: boolean read FMidTestError write FMidTestError;
    property MidTestErrorValue: double read FMidTestErrorValue write
FMidTestErrorValue;
    property Options: string read FOptions write FOptions;
    property SourceFileName: TFileName read FSourceFileName write
FSourceFileName;
    property TestAsValid: boolean read FTestAsValid write FTestAsValid;
    property TeachIdent: boolean read FTeachIdent write FTeachIdent;
    property TeachIdentCount: integer read FTeachIdentCount write
SetTeachIdentCount;
    property TestIdent: boolean read FTestIdent write FTestIdent;
    property TestIdentCount: integer read FTestIdentCount write FTestIdentCount;
    property UseForTeach: integer read FUseForTeach write SetUseForTeach;
  end;


procedure Register;

implementation

{$R *.RES}

{ TNeuron }

constructor TNeuron.Create(ALayer: TLayer);
begin
  inherited Create;
  // покажчик на шар у якому перебуває нейрон
  Layer := ALayer;
end;

destructor TNeuron.Destroy;
begin
  WeightCount := 0;
  FWeights := nil;
  Layer := nil;
```

```
  inherited;
end;

procedure TNeuron.ComputeOut(const AInputs: TVectorFloat);
var
  i: integer;
begin
  FOutput := 0;
  // Підраховується зважена сума нейрона
  for i := Low(AInputs) to High(AInputs) do
    FOutput := FOutput + FWeights[i] * AInputs[i];
end;

function TNeuron.GetWeights(Index: integer): double;
begin
  try
    Result := FWeights[Index];
  except
    on E: ERangeError do
      raise E.CreateFmt(SWeightRangeIndex, [Index])
  end;
end;

procedure TNeuron.InitWeights;
var
  i: integer;
begin
  // Ініціалізація ваг нейрона
  for i := Low(FWeights) to High(FWeights) do
    FWeights[i] := Random
end;

procedure TNeuron.SetWeightCount(const Value: integer);
begin
  SetLength(FWeights, Value);
end;

procedure TNeuron.SetWeights(Index: integer; Value: double);
begin
  try
    FWeights[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SWeightRangeIndex, [Index])
  end;
end;

{ Кінець опису TNeuron }

{ TNeuronHopf }

procedure TNeuronHopf.ComputeOut(const AInputs: TVectorFloat);
begin
  inherited;
  // гранична функція
  if FOutput >= 0 then
    FOutput := 1
  else
    FOutput := -1
end;

{ Кінець опису TNeuronHopf}

{ TNeuronBP }

destructor TNeuronBP.Destroy;
begin
  FOnActivation := nil;
  FOnActivation := nil;
```

```
  PrevUpdateCount := 0;
  FPrevUpdate := nil;
  inherited;
end;

function TNeuronBP.GetLearningRate(Index: integer): double;
begin
  Result := FLearningRate[Index];
end;

function TNeuronBP.GetPrevDerivative(Index: integer): double;
begin
  Result := FPrevDerivative[Index];
end;

function TNeuronBP.GetPrevUpdateCount: integer;
begin
  Result := High(FPrevUpdate) + 1;
end;

function TNeuronBP.GetPrevUpdate(Index: integer): double;
begin
  Result := FPrevUpdate[Index];
end;

procedure TNeuronBP.ComputeOut(const AInputs: TVectorFloat);
begin
  inherited;
  // Задає зсув нейрона
  FOutput := FOutput + Weights[High(AInputs) + 1];
  FOutput := OnActivation(FOutput);
end;

procedure TNeuronBP.SetDelta(Value: double);
begin
  FDelta := Value;
end;

procedure TNeuronBP.SetLearningRate(Index: integer; const Value: double);
begin
  FLearningRate[Index] := Value;
end;

procedure TNeuronBP.SetLearningRateCount(const Value: integer);
begin
  SetLength(FLearningRate, Value)
end;

procedure TNeuronBP.SetPrevUpdate(Index: integer; Value: double);
begin
  FPrevUpdate[Index] := Value;
end;

procedure TNeuronBP.SetPrevUpdateCount(const Value: integer);
begin
  SetLength(FPrevUpdate, Value)
end;

procedure TNeuronBP.SetPrevDerivative(Index: integer; const Value: double);
begin
  FPrevDerivative[Index] := Value;
end;

procedure TNeuronBP.SetPrevDerivativeCount(const Value: integer);
begin
  SetLength(FPrevDerivative, Value)
end;

{ Кінець опису TNeuronBP }
```

```
{ TLayer }

procedure TLayer.Assign(Source: TPersistent);
var
  i: integer;
begin
  FNumber := (Source as TLayer).FNumber;
  NeuronCount := (Source as TLayer).NeuronCount;
  // Створюються нейрони
  for i := 0 to NeuronCount - 1 do
    FNeurons[i] := TNeuron.Create(Self);
end;

constructor TLayer.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  inherited Create;
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuron.Create(Self);
end;

destructor TLayer.Destroy;
var
  i: integer;
begin
  for i := 0 to NeuronCount - 1 do
    FNeurons[i].Free;
  NeuronCount := 0;
  FNeurons := nil;
  inherited;
end;

function TLayer.GetNeuronCount: integer;
begin
  Result := High(FNeurons) + 1;
end;

function TLayer.GetNeurons(Index: integer): TNeuron;
begin
  Result := FNeurons[Index];
end;

procedure TLayer.SetNeuronCount(Value: integer);
begin
  if Value <> High(FNeurons) + 1 then
    SetLength(FNeurons, Value);
end;

procedure TLayer.SetNeurons(Index: integer; Value: TNeuron);
begin
  try
    FNeurons[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SNeuronRangeIndex, [Index])
  end;
end;

{ TLayerHopf }

constructor TLayerHopf.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  FNumber := ALayerNumber;
```

```
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuronHopf.Create(Self);
end;

{ TLayerBP }

procedure TLayerBP.Assign(Source: TPersistent);
var
  i: integer;
begin
  FNumber := (Source as TLayerBP).FNumber;
  NeuronCount := (Source as TLayerBP).NeuronCount;
  for i := 0 to NeuronCount - 1 do
    FNeurons[i] := TNeuronBP.Create(Self);
end;

constructor TLayerBP.Create(ALayerNumber: integer; ANeuronCount: integer);
var
  i: integer;
begin
  FNumber := ALayerNumber;
  NeuronCount := ANeuronCount;
  for i := 0 to ANeuronCount - 1 do
    FNeurons[i] := TNeuronBP.Create(Self);
end;

destructor TLayerBP.Destroy;
begin
  inherited;
end;

function TLayerBP.GetNeuronsBP(Index: integer): TNeuronBP;
begin
  Result := FNeurons[Index] as TNeuronBP;
end;

procedure TLayerBP.SetNeuronsBP(Index: integer; Value: TNeuronBP);
begin
  FNeurons[Index] := Value as TNeuronBP;
end;

{ TNeuralNet }

destructor TNeuralNet.Destroy;
begin
  Clear;
  SetLength(FPatternsInput, 0, 0);
  FPatternsInput := nil;
  SetLength(FPatternsOutput, 0 ,0);
  FPatternsOutput := nil;
  FLayers := nil;
  inherited;
end;

procedure TNeuralNet.Clear;
var
  i, xCount: integer;
begin
  xCount := LayerCount;
  if  xCount > 0 then
  begin
    for i := 0 to xCount - 1 do
      FLayers[i].Free;
    LayerCount := 0;
  end;
end;

function TNeuralNet.GetInputNeuronCount: integer;
```

```
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TNeuralNet.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TNeuralNet.GetLayers(Index: integer): TLayer;
begin
  Result := FLayers[Index];
end;

function TNeuralNet.GetOutputNeuronCount: integer;
begin
  Result := Layers[LayerCount - 1].NeuronCount;
end;

function TNeuralNet.GetPatternsInput(PatternIndex: integer; InputIndex:
integer): double;
begin
  Result := FPatternsInput[PatternIndex, InputIndex];
end;

procedure TNeuralNet.AddPattern(const AInputs: TVectorFloat; const AOutputs:
TVectorFloat);
var
  i: integer;
begin
  if InputNeuronCount <> High(AInputs) + 1 then
    raise EInOutDimensionError.Create(SInVectorCount);
  if OutputNeuronCount <> High(AOutputs) + 1 then
    raise EInOutDimensionError.Create(SOutVectorCount);
  PatternCount := PatternCount + 1;
  ResizeInputDim;
  ResizeOutputDim;
  for i := Low(AInputs) to High(AInputs) do
    PatternsInput[PatternCount - 1, i] := AInputs[i];
  for i := Low(AOutputs) to High(AOutputs) do
    PatternsOutput[PatternCount - 1, i] := AOutputs[i];
end;

procedure TNeuralNet.DeletePattern(Index: integer);
var
  i, j: integer;
begin
  try
    // видаляє вхідні значення приклада Index
    for i := Index to FPatternCount - 2 do
      for j := 0 to InputNeuronCount - 1 do
        FPatternsInput[i, j] := FPatternsInput[i + 1, j];
    // видаляє вихідні значення приклада Index
    for i := Index to FPatternCount - 2 do
      for j := 0 to OutputNeuronCount - 1 do
        FPatternsOutput[i, j] := FPatternsOutput[i + 1, j];
    Dec(FPatternCount);
    ResizeInputDim;
    ResizeOutputDim;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNet.ResetPatterns;
begin
  FPatternCount := DefaultPatternCount;
  ResizeInputDim;
```

```
  ResizeOutputDim;
end;

procedure TNeuralNet.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
  ResizeInputDim;
  ResizeOutputDim;
end;

procedure TNeuralNet.SetPatternsOutput(PatternIndex: integer; InputIndex:
integer; Value: double);
begin
  FPatternsOutput[PatternIndex, InputIndex] := Value;
end;

procedure TNeuralNet.SetPatternsInput(PatternIndex: integer; InputIndex:
integer; Value: double);
begin
  FPatternsInput[PatternIndex, InputIndex] := Value;
end;

procedure TNeuralNet.Init(const ANeuronsInLayer: TVectorInt);
var
  i,j: integer;
begin
  LayerCount := High(ANeuronsInLayer) + 1;
  // FLayers[0] нульовий шар і виконує роль розподільного,
  // використовується тільки поле Output
  FLayers[0] := TLayer.Create(0, ANeuronsInLayer[0]);
  // для нульового шару не потрібні вагові коефіцієнти
  for i := 1 to LayerCount - 1 do
  begin
    FLayers[i] := TLayer.Create(i, ANeuronsInLayer[i]);
    for j := 0 to ANeuronsInLayer[i] - 1 do
    with FLayers[i].FNeurons[j] do
      // задає кількість елементів у векторі ваг нейрона j в
      //  шарі i рівним кількості виходів попереднього шару
      WeightCount := FLayers[ i-1].NeuronCount;
  end;
end;

procedure TNeuralNet.ResizeInputDim;
begin
  SetLength(FPatternsInput, FPatternCount, InputNeuronCount)
end;

procedure TNeuralNet.ResizeOutputDim;
begin
  SetLength(FPatternsOutput, FPatternCount, OutputNeuronCount)
end;

procedure TNeuralNet.SetLayerCount(Value: integer);
begin
  SetLength(FLayers, Value);
end;

procedure TNeuralNet.SetLayers(Index: integer; Value: TLayer);
begin
  try
    FLayers[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;
```

```
{  TNeuralNetHopf }

constructor TNeuralNetHopf.Create(AOwner: TComponent);
begin
  inherited;
  PatternCount := DefaultPatternCount;
  InputNeuronCount := DefaultNeuronCount;
  MaxIterCount := DefaultMaxIterCount;
  AutoInit := False;
end;

destructor TNeuralNetHopf.Destroy;
begin
  FOnAfterInit := nil;
  FOnBeforeInit := nil;
  FOnPatternRecognized := nil;
  SetLength(FPatterns, 0, 0);
  FPatterns := nil;
  inherited;
end;

function TNeuralNetHopf.GetInput(Index: integer): double;
begin
  Result := Layers[1].Neurons[Index].Output;
end;

function TNeuralNetHopf.GetInputNeuronCount: integer;
begin
  Result := Layers[SensorLayer].NeuronCount;
end;

function TNeuralNetHopf.GetLayerCount: integer;
begin
  Result := DefaultHopfLayerCount;
end;

function TNeuralNetHopf.GetPatterns(InputIndex: integer; PatternIndex: integer):
integer;
begin
  Result := FPatterns[InputIndex, PatternIndex];
end;

function TNeuralNetHopf.Stabled: boolean;
var
  i: integer;
begin
  // Порівнює вихідні значення попередньої
  //  ітерації зі значеннями поточної
  Result := True;
  for i := 0 to InputNeuronCount - 1  do
    if FLayers[1].FNeurons[i].FOutput <> FLayers[0].FNeurons[i].FOutput then
    begin
      Result := False;
      Exit
    end;
end;

procedure TNeuralNetHopf.AddPattern(const ANewPattern: TVectorInt);
var
  i: integer;
begin
  if InputNeuronCount <> High(ANewPattern)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  PatternCount := PatternCount + 1;
  ResizePatternsDim;
  for i := 0 to FInputNeuronCount - 1 do
    FPatterns[FPatternCount - 1, i] := ANewPattern[i];
  if AutoInit then
```

```
    InitWeights;
end;

procedure TNeuralNetHopf.Calc;
var
  i: integer;
  xCurrentIter: integer;
  xArray: TVectorFloat;
begin
  SetLength(xArray, InputNeuronCount);
  // Цикл працює поки не стабілізуються виходи
  xCurrentIter := 0;
  repeat
    for i := 0 to InputNeuronCount - 1 do
    begin
      // Запам'ятовує попередній крок ітерації, для
      //  цього використовується нульовий шар
      Layers[SensorLayer].Neurons[i].Output := Layers[1].Neurons[i].Output;
      xArray[i] := Layers[1].Neurons[i].Output;
    end;
    for i := 0 to InputNeuronCount - 1 do
      with Layers[1].Neurons[i] do
        // Розраховується новий стан нейронів і аксонів
        ComputeOut(xArray);
    Inc(xCurrentIter);
  until Stabled or (MaxIterCount = xCurrentIter);
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetHopf.DeletePattern(Index: integer);
var
  i, j: integer;
begin
  try
    for i := Index to FPatternCount - 2 do
      for j := 0 to FInputNeuronCount - 1 do
        FPatterns[i, j] := FPatterns[i + 1, j];
    Dec(FPatternCount);
    ResizePatternsDim;
    if AutoInit then
      InitWeights;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNetHopf.Init;
var
  i, j: integer;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
  LayerCount := DefaultHopfLayerCount;
  for i := 0 to LayerCount - 1 do
    FLayers[i] := TLayerHopf.Create(i, FInputNeuronCount);
  // Для нульового шару не потрібні вагові коефіцієнти
  for j := 0 to FInputNeuronCount - 1 do
    with FLayers[1].FNeurons[j] do
      // задає кількість елементів у векторі
      WeightCount := FInputNeuronCount;
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TNeuralNetHopf.InitWeights;
```

```
var
  i, j, k : integer;
begin
  // Ініціалізує вагову матрицю
  for i := 0 to InputNeuronCount - 1 do
    for j := 0 to InputNeuronCount - 1 do
    with Layers[1].Neurons[i] do
    begin
      Weights[j] := 0;
      if i <> j then
        for k := 0 to PatternCount - 1 do
          Weights[j] := Weights[j]+ Patterns[k, i] * Patterns[k, j]
    end;
end;

procedure TNeuralNetHopf.ResetPatterns;
begin
  PatternCount := DefaultPatternCount;
  if AutoInit then
    InitWeights;
end;

procedure TNeuralNetHopf.ResizePatternsDim;
begin
  SetLength(FPatterns, FPatternCount, FInputNeuronCount);
end;

procedure TNeuralNetHopf.SetInput(Index: integer; Value: double);
begin
  try
    Layers[1].Neurons[Index].Output := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SPatternRangeIndex, [Index])
  end;
end;

procedure TNeuralNetHopf.SetInputNeuronCount(Value: integer);
begin
  if Value > DefaultNeuronCount then
    FInputNeuronCount := Value
  else
    FInputNeuronCount := DefaultNeuronCount;
  ResizePatternsDim;
  Init;
end;

procedure TNeuralNetHopf.SetPatternCount(const Value: integer);
begin
  if Value < DefaultPatternCount then
    FPatternCount := DefaultPatternCount
  else
    FPatternCount := Value;
end;

procedure TNeuralNetHopf.SetPatterns(InputIndex: integer; PatternIndex: integer;
Value: integer);
begin
  FPatterns[InputIndex, PatternIndex] := Value;
end;

{ TNeuralNetBP }

constructor TNeuralNetBP.Create(AOwner: TComponent);
var
  i: integer;
begin
  inherited;
  FNeuronsInLayer := TStringList.Create;
```

```
  for i := 0 to DefaultLayerCount do
    AddLayer(DefaultNeuronCount);
  TStringList(FNeuronsInLayer).OnChange := NeuronsInLayerChange;
  AutoInit := True;
  StopTeach := False;
  TeachStopped := False;
  NeuronsInLayerChange(Self);
  SetDefaultProperties;
end;

destructor TNeuralNetBP.Destroy;
begin
  FNeuronsInLayer.Free;
  SetLength(FRandomOrder, 0);
  FRandomOrder := nil;
  SetLength(FDesiredOut, 0);
  FDesiredOut := nil;
  SetLength(FTestSetPatterns, 0, 0);
  FTestSetPatterns := nil;
  SetLength(FTestSetPatternsOut, 0, 0);
  FTestSetPatternsOut := nil;
  FOnAfterInit := nil;
  FOnAfterTeach := nil;
  FOnBeforeInit := nil;
  FOnBeforeTeach := nil;
  FOnEpochPassed := nil;
  inherited;
end;

function TNeuralNetBP.GetLayersBP(Index: integer): TLayerBP;
begin
  Result := FLayers[Index] as TLayerBP;
end;

function TNeuralNetBP.GetLayerCount: integer;
begin
  Result := High(FLayers) + 1;
end;

function TNeuralNetBP.GetDesiredOut(Index: integer): double;
begin
  Result := FDesiredOut[Index];
end;

function TNeuralNetBP.GetOutput(Index: integer): double;
begin
  try
    Result := LayersBP[LayerCount - 1].NeuronsBP[Index].Output;
  except
    on E: ERangeError do
      raise E.CreateFmt(SNeuronRangeIndex, [Index])
  end;
end;

function TNeuralNet.GetPatternsOutput(PatternIndex: integer; OutputIndex:
integer): double;
begin
  Result := FPatternsOutput[PatternIndex, OutputIndex];
end;

function TNeuralNetBP.QuadError: double;
var
  i: integer;
begin
  // розраховує середньоквадратичну помилку
  Result := 0;
  for i := 0 to OutputNeuronCount - 1 do
    Result := Result + sqr(LayersBP[LayerCount - 1].NeuronsBP[i].Output -
DesiredOut[i]);
```

```
  Result := Result/2;
end;

function TNeuralNetBP.Activation(Value: double): double;
begin
  // Активаційна функція - сигмоїд
  Result := 1/( 1 + exp(-FAlpha * Value) )
end;

function TNeuralNetBP.Activation(Value: double): double;
begin
  // Похідна сигмоїди
  Result := FAlpha * Value * (1 - Value)
end;

function TNeuralNetBP.GetTestSetPatterns(InputIndex, PatternIndex: integer):
double;
begin
  Result := FTestSetPatterns[InputIndex, PatternIndex];
end;

function TNeuralNetBP.GetTestSetPatternsOut(InputIndex, PatternIndex: integer):
double;
begin
  Result := FTestSetPatternsOut[InputIndex, PatternIndex];
end;

procedure TNeuralNetBP.AddLayer(ANeurons: integer);
begin
  if ANeurons < DefaultNeuronCount then
    NeuronCountError
  else
    NeuronsInLayer.Add(IntToStr(ANeurons));
end;

procedure TNeuralNetBP.AdjustWeights;
var
  i, j, k: integer;
  xCurrentUpdate: double;
begin
  // Підстроювання ваг починаючи з першого шару
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
    begin
      for k := 0 to LayersBP[ i-1].NeuronCount  do
      with LayersBP[i].NeuronsBP[j] do
      begin
        // коректує вагу з'єднуючого j-нейрона шару i
        //  з k-нейроном шару i-1:  добутком дельта j-нейрона
        //  на вихід k-нейрона шару i-1
        if k = LayersBP[ i-1].NeuronCount then
           // якщо це нейрон, що задає зсув
           xCurrentUpdate := -TeachRate * Delta + Momentum * PrevUpdate[k]
        else
           xCurrentUpdate := -TeachRate * Delta *
           LayersBP[ i-1].NeuronsBP[k].Output + Momentum * PrevUpdate[k];
        Weights[k]:= Weights[k] + xCurrentUpdate;
        PrevUpdate[k] := xCurrentUpdate;
      end;
    end
end;

procedure TNeuralNetBP.CalcLocalError;
var
  i, j, k: integer;
begin
  // Дельта-правило з останнього шару до першого
  for i := LayerCount - 1 downto 1 do
    // для останнього шару
```

```
    if i = LayerCount - 1 then
      for j := 0 to LayersBP[i].NeuronCount - 1 do
        LayersBP[i].NeuronsBP[j].Delta := (LayersBP[i].NeuronsBP[j].Output-
DesiredOut[j])
        * Activation(LayersBP[i].NeuronsBP[j].Output)
    else
      for j := 0 to LayersBP[i].NeuronCount - 1 do
        with LayersBP[i].NeuronsBP[j] do
        begin
          Delta := 0;
          // Підсумує добуток локальної помилки k-нейрона шару i+1
          //  на вагу з'єднуючий k-нейрон шару i+1 з j-нейроном шару i
          for k := 0 to LayersBP[i+1].NeuronCount - 1 do
            Delta := Delta + LayersBP[i+1].NeuronsBP[k].Delta *
                             LayersBP[i+1].NeuronsBP[k].Weights[j];
          Delta := Delta * Activation(Output)
        end;
end;

procedure TNeuralNetBP.CheckTestSet;
var
  i, j: integer;
  xArray: TVectorFloat;
  xFirstTestSample: boolean;
  xQuadError: double;
  // функція розраховує середньоквадратичну помилку
  function QuadError(APatternCount: integer): double;
  var
    i: integer;
  begin
    Result := 0;
    for i := 0 to OutputNeuronCount - 1 do
      Result := Result + sqr(LayersBP[LayerCount - 1].NeuronsBP[i].Output -
TestSetPatternsOut[APatternCount, i]);
    Result := Result/2;
  end;
begin
  SetLength(xArray, InputNeuronCount);
  xFirstTestSample := True;
  FRecognizedTestCount := 0;
  FMidTestResidual := 0;
  FMaxTestResidual := 0;
  for i := 0 to TestSetPatternCount - 1 do
  begin
    for j := 0 to InputNeuronCount - 1 do
      xArray[j] := TestSetPatterns[i, j];
    Compute(xArray);
    xQuadError := QuadError(i);
    // перевірка - чи розпізнаний приклад з тестової множини
    if xQuadError < IdentError then
      Inc(FRecognizedTestCount);
    FMidTestResidual := FMidTestResidual + xQuadError;
    // максимальна помилка на тестовій множині
    if xFirstTestSample then
    begin
      FMaxTestResidual := xQuadError;
      xFirstTestSample := False;
    end
    else
      if FMaxTestResidual < xQuadError then
        FMaxTestResidual := xQuadError;
  end;
  // середня помилка на тестовій множині
  FMidTestResidual := FMidTestResidual/TestSetPatternCount;
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetBP.Compute(AVector: TVectorFloat);
```

```
var
  i: integer;
begin
  if InputNeuronCount <> High(AVector)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  for i := Low(AVector) to High(AVector) do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  AVector[i];
  Propagate;
end;

procedure TNeuralNetBP.DoOnAfterInit;
begin
  if Assigned(FOnAfterInit) then
    FOnAfterInit(Self);
end;

procedure TNeuralNetBP.DoOnAfterNeuronCreated(ALayerIndex, ANeuronIndex:
integer);
var
  i: integer;
begin
  with LayersBP[ALayerIndex].NeuronsBP[ANeuronIndex] do
    for i := 0 to PrevUpdateCount - 1 do
      PrevUpdate[i] := 0;
  if Assigned(FOnAfterNeuronCreated) then
    FOnAfterNeuronCreated(Self);
end;

procedure TNeuralNetBP.DoOnAfterTeach;
begin
  if Assigned(FOnAfterTeach) then
    FOnAfterTeach(Self);
end;

procedure TNeuralNetBP.DoOnBeforeInit;
begin
  if Assigned(FOnBeforeInit) then
    FOnBeforeInit(Self);
end;

procedure TNeuralNetBP.DoOnBeforeTeach;
begin
  if Assigned(FOnBeforeTeach) then
    FOnBeforeTeach(Self);
end;

procedure TNeuralNetBP.DoOnEpochPassed;
begin
  if Assigned(FOnEpochPassed) then
    FOnEpochPassed(Self);
end;

procedure TNeuralNetBP.DeleteLayer(Index: integer);
var
  i: integer;
begin
  try
    NeuronsInLayer.Delete(Index);
    for i := Index to LayerCount - 2 do
      LayersBP[i].Assign(LayersBP[i + 1]);
    FLayers[LayerCount - 1].Free;
    LayerCount := LayerCount - 1;
  except
    on E: ERangeError do
      raise E.CreateFmt(SLayerRangeIndex, [Index])
  end;
end;

procedure TNeuralNetBP.Init;
```

```
var
  i, j: integer;
begin
  DoOnBeforeInit;
  if NeuronsInLayer.Count > 0 then
  begin
  LayerCount := NeuronsInLayer.Count;
    // FLayers[0] нульовий шар, використовується тільки поле Output
    FLayers[0] := TLayerBP.Create(0, StrToInt(NeuronsInLayer.Strings[0]));
    // для нульового шару не потрібні вагові коефіцієнти
    for i := 1 to LayerCount - 1 do
    begin
      FLayers[i] := TLayerBP.Create(i, StrToInt(NeuronsInLayer.Strings[i]));
      for j := 0 to StrToInt(NeuronsInLayer.Strings[i]) - 1 do
      with LayersBP[i].NeuronsBP[j] do
      begin
        // задає кількість елементів у векторі ваг + зсув
        WeightCount := LayersBP[ i-1].NeuronCount + BiasNeuron;
        // задає кількість  у векторі утримуючих попередню
        //  корекцію елементів  + зсув
        PrevUpdateCount := LayersBP[ i-1].NeuronCount + BiasNeuron;
        PrevDerivativeCount := LayersBP[ i-1].NeuronCount + BiasNeuron; // для
швидких алгоритмів
        LearningRateCount := LayersBP[ i-1].NeuronCount + BiasNeuron;   // для
швидких алгоритмів
        OnActivation := Activation;
        OnActivation := Activation;
        Randomize;
        DoOnAfterNeuronCreated(i, j);
      end
    end;
    // установлює розмірність масиву виходів
    //  число нейронів в останньому шарі = числу виходів
    SetLength(FDesiredOut, OutputNeuronCount);
  end;
  DoOnAfterInit;
end;

procedure TNeuralNetBP.InitWeights;
var
  i,j: integer;
begin
  Randomize;
  // Ініціалізація ваг
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
       LayersBP[i].NeuronsBP[j].InitWeights;
end;

procedure TNeuralNetBP.LoadPatternsInput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to InputNeuronCount - 1 do
    LayersBP[SensorLayer].NeuronsBP[i].Output :=  PatternsInput[APatternIndex,
i];
end;

procedure TNeuralNetBP.LoadPatternsOutput(APatternIndex :integer);
var
  i: integer;
begin
  for i := 0 to OutputNeuronCount - 1 do
    DesiredOut[i] := PatternsOutput[APatternIndex, i];
end;

procedure TNeuralNetBP.NeuronsInLayerChange(Sender: TObject);
begin
  if AutoInit then
```

```
    Init;
end;

procedure TNeuralNetBP.NeuronCountError;
begin
  raise ENeuronCountError.Create(SNeuronCount)
end;

procedure TNeuralNetBP.Propagate;
var
  i, j, xIndex: integer;
  xArray: TVectorFloat;
begin
   // Поширення сигналу в прямому напрямку з першого шару
  for i := 1 to LayerCount - 1 do
  begin
    // формування масиву входів з виходів попереднього шару
    SetLength(xArray, LayersBP[ i-1].NeuronCount);
    for xIndex := 0 to LayersBP[ i-1].NeuronCount - 1 do
       xArray[xIndex] := LayersBP[ i-1].NeuronsBP[xIndex].Output;
    // обчислення виходу нейрона
    for j := 0 to LayersBP[i].NeuronCount - 1 do
      with LayersBP[i].NeuronsBP[j] do
        ComputeOut(xArray);
    for xIndex := 0 to LayersBP[ i-1].NeuronCount - 1 do
       xArray[xIndex] := 0;
  end;
  SetLength(xArray, 0);
  xArray := nil;
end;

procedure TNeuralNetBP.ResetLayers;
begin
  Clear;
  FNeuronsInLayer.Clear;
end;

procedure TNeuralNetBP.SetDesiredOut(Index: integer; Value: double);
begin
  FDesiredOut[Index] := Value;
end;

procedure TNeuralNetBP.SetLayersBP(Index: integer; Value: TLayerBP);
begin
  FLayers[Index] := Value as TLayerBP;
end;

procedure TNeuralNetBP.SetAlpha(Value: double);
begin
  if (Value > 10) or (Value < 0.01) then
    FAlpha := DefaultAlpha
  else
    FAlpha := Value;
end;

procedure TNeuralNetBP.SetTeachRate(Value: double);
begin
  if (Value > 1) or (Value <= 0) then
    FTeachRate := DefaultTeachRate
  else
    FTeachRate := Value;
end;

procedure TNeuralNetBP.SetTestSetPatterns(InputIndex, PatternIndex: integer;
const Value: double);
begin
  FTestSetPatterns[InputIndex, PatternIndex] := Value;
end;
```

```
procedure TNeuralNetBP.SetTestSetPatternsOut(InputIndex, PatternIndex: integer;
const Value: double);
begin
  FTestSetPatternsOut[InputIndex, PatternIndex] := Value;
end;

procedure TNeuralNetBP.SetTestSetPatternCount(const Value: integer);
begin
  FTestSetPatternCount := Value;
  SetLength(FTestSetPatterns, FTestSetPatternCount, InputNeuronCount);
  SetLength(FTestSetPatternsOut, FTestSetPatternCount, OutputNeuronCount);
end;

procedure TNeuralNetBP.SetMomentum(Value: double);
begin
  if (Value > 1) or (Value < 0) then
    FMomentum := DefaultMomentum
  else
    FMomentum := Value;
end;

procedure TNeuralNetBP.SetEpochCount(Value: integer);
begin
  if Value < 1 then
    FEpochCount := 1
  else
    FEpochCount := Value;
end;

procedure TNeuralNetBP.ShakeUp;
var
  i, j, k: integer;
begin
  Randomize;
  for i := 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
      for k := 0 to LayersBP[ i-1].NeuronCount  do
        with LayersBP[i].NeuronsBP[j] do
          Weights[k]:= Weights[k] + Random*0.1-0.05;
end;

procedure TNeuralNetBP.Shuffle;
var
  i, j, xNewInd, xLast: integer;
  xIsUnique : boolean;
begin
  xNewInd := 0;
  FRandomOrder[0] := Round(Random(FPatternCount));
  xLast := 0;
  for i := 1 to PatternCount - 1 do
  begin
    xIsUnique := False;
    while not xIsUnique do
    begin
      xNewInd := Round((Random(FPatternCount)));
      xIsUnique := True;
      for j := 0 to xLast do
        if xNewInd = FRandomOrder[j] then
          xIsUnique := False;
    end;
    FRandomOrder[i] := xNewInd;
    xLast := xLast +1;
  end;
end;

procedure TNeuralNetBP.TeachOffLine;
var
  j: integer;
  xQuadError: double;
```

```
  xNewEpoch: boolean;
begin
  DoOnBeforeTeach;
  if not ContinueTeach then
  begin
    // ваги ініціалізуються, якщо мережа навчається з "нуля"
    InitWeights;
    FEpochCurrent := 1;
  end;
  Randomize;
  SetLength(FRandomOrder, FPatternCount);
  TeachStopped := False;
  while (FEpochCurrent <= EpochCount) do
  begin
    FTeachError := 0;
    FMaxTeachResidual := 0;
    FRecognizedTeachCount := 0;
    xNewEpoch := True;
    Shuffle;
    for j := 0 to PatternCount - 1 do
    begin
      LoadPatternsInput(FRandomOrder[j]);
      LoadPatternsOutput(FRandomOrder[j]);
      Propagate;
      xQuadError := QuadError;
      // перевірка - чи розпізнаний приклад з навчальної множини
      if xQuadError < IdentError then
        Inc(FRecognizedTeachCount);
      FTeachError := FTeachError + xQuadError;
      // максимальна помилка на навчальній множині
      if xNewEpoch then
      begin
        FMaxTeachResidual := xQuadError;
        xNewEpoch := False;
      end
      else
        if MaxTeachResidual < xQuadError then
          FMaxTeachResidual := xQuadError;
      CalcLocalError;
      AdjustWeights;
    end;
    // середня помилка на навчальній множині
    FMidTeachResidual := TeachError/PatternCount;
    // перевірка мережі на узагальнення
    if TestSetPatternCount > 0 then
      CheckTestSet;
    DoOnEpochPassed;
    if StopTeach then
    begin
      TeachStopped := True;
      Exit;
    end;
    Inc(FEpochCurrent);
  end;
  DoOnAfterTeach;
end;

procedure TNeuralNetBP.SetPatternCount(const Value: integer);
begin
  FPatternCount := Value;
  inherited;
end;

procedure TNeuralNetBP.SetDefaultProperties;
begin
  // параметри встановлювані за замовчуванням
  Alpha := DefaultAlpha;
  ContinueTeach := False;
  Epoch := True;
```

```
  EpochCount := DefaultEpochCount;
  Momentum := DefaultMomentum;
  TeachRate := DefaultTeachRate;
  ResizeInputDim;
  ResizeOutputDim;
end;

{ TNeuralNetExtended }

constructor TNeuralNetExtended.Create(AOwner: TComponent);
begin
  inherited;
  SetDefaultProperties;
end;

destructor TNeuralNetExtended.Destroy;
var
  i: integer;
begin
  if Assigned(FNnwFile) then
    FNnwFile.Free;
  FNeuroDataSource.Free;
  for i := 0 to FAvailableFieldsCount - 1 do
    FFields[i].Free;
  inherited;
end;

function TNeuralNetExtended.GetFields(Index: integer): TNeuroField;
begin
  Result := FFields[Index];
end;

function TNeuralNetExtended.GetInputFieldCount: integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if Fields[i].KindName = fdInput then
      Inc(Result);
end;

function TNeuralNetExtended.GetOutputFieldCount: integer;
var
  i: integer;
begin
  Result := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if Fields[i].KindName = fdOutput then
      Inc(Result);
end;

function TNeuralNetExtended.GetOutput(Index: integer): double;
var
  xTmp: double;
begin
  with Fields[RealOutputIndex[Index]] do
    case NormTypeName of
      nrmAuto: begin
                 xTmp := -ln(1/LayersBP[LayerCount - 1].NeuronsBP[Index].Output -
1);
                 LayersBP[LayerCount - 1].NeuronsBP[Index].Output := xTmp *
Dispersion + ValueMid;
               end;
      nrmLinear:  Result := (LayersBP[LayerCount - 1].NeuronsBP[Index].Output +
1)*(ValueMax - ValueMin)/2 + ValueMin;
      nrmLinearOut: Result := LayersBP[LayerCount -
1].NeuronsBP[Index].Output*(ValueMax - ValueMin) + ValueMin;
```

```
      nrmSigmoid:  Result := - Ln(1/LayersBP[LayerCount -
1].NeuronsBP[Index].Output - 1)/Alpha;
    end;
end;

function TNeuralNetExtended.GetRealInputIndex(Index: integer): integer;
begin
  Result := FRealInputIndex[Index];
end;

function TNeuralNetExtended.GetRealOutputIndex(Index: integer): integer;
begin
  Result := FRealOutputIndex[Index];
end;

procedure TNeuralNetExtended.ComputeUnPrepData(AVector: TVectorFloat);
var
  i: integer;
  xTmp: double;
begin
  if InputNeuronCount <> High(AVector)+ 1 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  for i := Low(AVector) to High(AVector) do
  with FFields[RealInputIndex[i]] do
    case NormTypeName of
      nrmAuto: begin
                 xTmp := (LayersBP[SensorLayer].NeuronsBP[i].Output -
ValueMid)/Dispersion;
                 LayersBP[SensorLayer].NeuronsBP[i].Output := 1/(1 + exp(-
xTmp));
               end;
      nrmLinear: LayersBP[SensorLayer].NeuronsBP[i].Output := 2*(AVector[i] -
ValueMin)/(ValueMax - ValueMin) - 1;
      nrmLinearOut: LayersBP[SensorLayer].NeuronsBP[i].Output := (AVector[i] -
ValueMin)/(ValueMax - ValueMin);
      nrmSigmoid: LayersBP[SensorLayer].NeuronsBP[i].Output := 1/( 1 + exp(-
Alpha * AVector[i]));
    end;
  Propagate;
end;

procedure TNeuralNetExtended.DoOnBeforeTeach;
begin
  if InputNeuronCount <> InputFieldCount then
    raise ENeuronNotEqualFieldError.Create(SInFieldCount);
  if OutputNeuronCount <> OutputFieldCount then
    raise ENeuronNotEqualFieldError.Create(SOutFieldCount);
  if InputNeuronCount < 0 then
    raise EInOutDimensionError.Create(SInNeuronCount);
  if OutputNeuronCount < 0 then
    raise EInOutDimensionError.Create(SOutNeuronCount);
  if (not FMaxTeachError) and (not FMaxTestError) and
    (not FMidTeachError) and (not FMidTestError) and (not FEpoch) then
    raise EBPStopCondition.Create(SBPStopCondition);
  inherited DoOnBeforeTeach;
end;

procedure TNeuralNetExtended.DoOnEpochPassed;
begin
  if MaxTeachError and (MaxTeachResidual < MaxTeachErrorValue) then
    StopTeach := True;
  if MidTeachError and (MidTeachResidual < MidTeachErrorValue) then
    StopTeach := True;
  if MaxTestError and (MaxTestResidual < MaxTestErrorValue) then
    StopTeach := True;
  if MidTestError and (MidTestResidual < MidTestErrorValue) then
    StopTeach := True;
  if TeachIdent and (Round((FRecognizedTeachCount * 100)/PatternCount) <
TeachIdentCount) then
```

```
    StopTeach := True;
  if TestIdent and (Round((FRecognizedTestCount * 100)/TestSetPatternCount) <
TestIdentCount) then
    StopTeach := True;
  inherited DoOnEpochPassed;
end;

procedure TNeuralNetExtended.LoadDataFrom;
var
  xTempStream: TFileStream;
  i, j: integer;
  xFieldCount: integer;
  xArray: TVectorFloat;
  xPatternsList: TStringList;
begin
  // створюється потік
  xTempStream := TFileStream.Create(FSourceFileName, fmOpenRead);
  // створюється список
  xPatternsList := TStringList.Create;
  xPatternsList.LoadFromStream(xTempStream);
  try
    if SettingsLoaded then
    begin
      xFieldCount := FNeuroDataSource.FieldCount(xPatternsList.Strings[0]);
      if AvailableFieldsCount <> xFieldCount then
        if MessageDlg('Кількість полів у файлі даних не відповідає значенню
AvailableFieldsCount'+ #13 + 'Установити нове значення AvailableFieldsCount = '+
IntToStr(xFieldCount),
                    mtConfirmation, [mbYes, mbNo], 0) = mrYes then
          AvailableFieldsCount := xFieldCount;
    end
    else
      AvailableFieldsCount :=
FNeuroDataSource.FieldCount(xPatternsList.Strings[0]);
    FNeuroDataSource.ExtractHeaders(FFields, xPatternsList.Strings[0]);
    // установлюється розмірність часового масиву
    SetLength(xArray, FAvailableFieldsCount);
    // установлюється розмірність масиву даних
    if FUseForTeach = 100 then
      PatternCount := xPatternsList.Count - 1
    else
    begin
      PatternCount := Round((xPatternsList.Count - 1) * FUseForTeach / 100);
      TestSetPatternCount := xPatternsList.Count - PatternCount - 1;
    end;
    for i := 0 to FAvailableFieldsCount - 1 do
      FFields[i].DataInCount := xPatternsList.Count - 1;
    for j := 0 to xPatternsList.Count - 2 do
    begin
      FNeuroDataSource.ExtractValues(xArray, xPatternsList.Strings[j + 1]);
      for i := 0 to FAvailableFieldsCount - 1 do
        FFields[i].DataIn[j] := xArray[i]
    end;
  finally
    xTempStream.Free;
    xPatternsList.Free;
    SetLength(xArray, 0);
    xArray := nil;
  end;
end;

procedure TNeuralNetExtended.LoadPhase1;
begin
  FSourceFileName := FNnwFile.ReadString('Phase1', 'LearnSampleFileName', '');
  FNeuroDataSource.Name := FSourceFileName;
end;

procedure TNeuralNetExtended.LoadPhase2;
var
```

```
  i: integer;
begin
  AvailableFieldsCount := FNnwFile.ReadInteger('Phase2', 'AvailableFieldsCount',
1);
  for i := 0 to AvailableFieldsCount - 1 do
  with FFields[i] do
  begin
    Name := FNnwFile.ReadString('Phase2', 'FieldName_'+IntToStr(i), '');
    Kind := FNnwFile.ReadInteger('Phase2', 'FieldType_'+IntToStr(i), 0);
    NormType := FNnwFile.ReadInteger('Phase2', 'NormType_'+IntToStr(i), 0);
    ValueMax := FNnwFile.ReadFloat('Phase2', 'Max_'+IntToStr(i), 0);
    ValueMin := FNnwFile.ReadFloat('Phase2', 'Min_'+IntToStr(i), 0);
    ValueMid := FNnwFile.ReadFloat('Phase2', 'Mid_'+IntToStr(i), 0);
    Dispersion := FNnwFile.ReadFloat('Phase2', 'Disp_'+IntToStr(i), 0);
    Alpha := FNnwFile.ReadFloat('Phase2', 'Alpha_'+IntToStr(i), 0);
    Ind := FNnwFile.ReadBool('Phase2', 'Ind_'+IntToStr(i), False);
  end;
  SettingsLoaded := True;
end;

procedure TNeuralNetExtended.LoadPhase4;
begin
  UseForTeach := FNnwFile.ReadInteger('Phase4', 'UseForTeach',
DefaultUseForTeach);
  IdentError:= FNnwFile.ReadFloat('Phase4', 'IdentErr', DefaultErrorValue);
  TestAsValid := FNnwFile.ReadBool('Phase4', 'TestAsValid', False);
  Epoch:= FNnwFile.ReadBool('Phase4', 'Epoch', False);
  EpochCount:= FNnwFile.ReadInteger('Phase4', 'Epoch', DefaultEpochCount);
  MaxTeachError:= FNnwFile.ReadBool('Phase4', 'MaxTeachErr', False);
  MaxTeachErrorValue:= FNnwFile.ReadFloat('Phase4', 'MaxTeachErr',
DefaultErrorValue);
  MidTeachError:= FNnwFile.ReadBool('Phase4', 'MidTeachErr', False);
  MidTeachErrorValue:= FNnwFile.ReadFloat('Phase4', 'MidTeachErr',
DefaultErrorValue);
  TeachIdent:= FNnwFile.ReadBool('Phase4', 'TeachIdent', False);
  TeachIdentCount:= FNnwFile.ReadInteger('Phase4', 'TeachIdent',
DefaultTeachIdentCount);
  MaxTestError:= FNnwFile.ReadBool('Phase4', 'MaxTestErr', False);
  MaxTestErrorValue:= FNnwFile.ReadFloat('Phase4', 'MaxTestErr',
DefaultErrorValue);
  MidTestError:= FNnwFile.ReadBool('Phase4', 'MidTestErr', False);
  MidTestErrorValue:= FNnwFile.ReadFloat('Phase4', 'MidTestErr',
DefaultErrorValue);
  TestIdent:= FNnwFile.ReadBool('Phase4', 'TestIdent', False);
  TestIdentCount:= FNnwFile.ReadInteger('Phase4', 'TestIdent',
DefaultTestIdentCount);
end;

procedure TNeuralNetExtended.LoadNetwork;
var
  i,j,k: integer;
  xLayerCount: integer;
begin
  // знищується поточна конфігурація нейромережі
  ResetLayers;
  Alpha := FNnwFile.ReadFloat('Network', 'Alpha', DefaultAlpha);
  Momentum := FNnwFile.ReadFloat('Network', 'Miu', DefaultMomentum);
  TeachRate := FNnwFile.ReadFloat('Network', 'TeachSpeed', DefaultTeachRate);
  EpochCount := FNnwFile.ReadInteger('Network', 'Epoch', DefaultEpochCount);
  xLayerCount := FNnwFile.ReadInteger('Network','CountLayers',
DefaultLayerCount);
  // задається кількість нейронів у шарах
  AutoInit := False;
  for i := 0 to xLayerCount - 1 do
    AddLayer(FNnwFile.ReadInteger('Network','Layer_'+IntToStr(i),
DefaultNeuronCount));
  AutoInit := True;
  // ініціалізація нової конфігурації нейромережі
  Init;
```

```
  // завантаження вагових коефіцієнтів і зсуву
  for i:= 1 to LayerCount - 1 do
    for j := 0 to LayersBP[i].NeuronCount - 1 do
    begin
      for k := 0 to LayersBP[ i-1].NeuronCount - 1 do
        LayersBP[i].NeuronsBP[j].Weights[k] := FNnwFile.ReadFloat('Network',
                              'W_'+IntToStr( i-
1)+'_'+IntToStr(k)+'_'+IntToStr(j), 0);
      LayersBP[i].NeuronsBP[j].Weights[LayersBP[ i-1].NeuronCount] :=
FNnwFile.ReadFloat('Network',
       'WT_'+IntToStr( i-1)+'_'+IntToStr(j), 0);
    end;
end;

procedure TNeuralNetExtended.SavePhase1;
begin
  FNnwFile.WriteString('Phase1', 'LearnSampleFileName', FNeuroDataSource.Name);
end;

procedure TNeuralNetExtended.SavePhase2;
var
  i: integer;
begin
  FNnwFile.WriteInteger('Phase2', 'AvailableFieldsCount',
FAvailableFieldsCount);
  for i := 0 to AvailableFieldsCount - 1 do
  with Fields[i] do
  begin
    FNnwFile.WriteString('Phase2', 'FieldName_'+IntToStr(i), Name);
    FNnwFile.WriteInteger('Phase2', 'FieldType_'+IntToStr(i), Kind);
    FNnwFile.WriteInteger('Phase2', 'NormType_'+IntToStr(i), NormType);
    FNnwFile.WriteFloat('Phase2', 'Max_'+IntToStr(i), ValueMax);
    FNnwFile.WriteFloat('Phase2', 'Min_'+IntToStr(i), ValueMin);
    FNnwFile.WriteFloat('Phase2', 'Mid_'+IntToStr(i), ValueMid);
    FNnwFile.WriteFloat('Phase2', 'Disp_'+IntToStr(i), Dispersion);
    FNnwFile.WriteFloat('Phase2', 'Alpha_'+IntToStr(i), Alpha);
    FNnwFile.WriteBool('Phase2', 'Ind_'+IntToStr(i), Ind);
  end;
end;

procedure TNeuralNetExtended.SavePhase4;
begin
  FNnwFile.WriteBool('Phase4', 'Epoch', Epoch);
  FNnwFile.WriteInteger('Phase4', 'Epoch', EpochCount);
  FNnwFile.WriteFloat('Phase4', 'IdentErr', IdentError);
  FNnwFile.WriteBool('Phase4', 'MaxTeachErr', MaxTeachError);
  FNnwFile.WriteFloat('Phase4', 'MaxTeachErr', MaxTeachErrorValue);
  FNnwFile.WriteBool('Phase4', 'MaxTestErr', MaxTestError);
  FNnwFile.WriteFloat('Phase4', 'MaxTestErr', MaxTestErrorValue);
  FNnwFile.WriteBool('Phase4', 'MidTeachErr', MidTeachError);
  FNnwFile.WriteFloat('Phase4', 'MidTeachErr', MidTeachErrorValue);
  FNnwFile.WriteBool('Phase4', 'MidTestErr', MidTestError);
  FNnwFile.WriteFloat('Phase4', 'MidTestErr', MidTestErrorValue);
  FNnwFile.WriteFloat('Phase4', 'Miu', Momentum);
  FNnwFile.WriteBool('Phase4', 'TeachIdent', TeachIdent);
  FNnwFile.WriteFloat('Phase4', 'TeachSpeed', TeachRate);
  FNnwFile.WriteInteger('Phase4', 'TeachIdent', TeachIdentCount);
  FNnwFile.WriteBool('Phase4', 'TestAsValid', TestAsValid);
  FNnwFile.WriteBool('Phase4', 'TestIdent', TestIdent);
  FNnwFile.WriteInteger('Phase4', 'TestIdent', TestIdentCount);
  FNnwFile.WriteInteger('Phase4', 'UseForTeach', UseForTeach);
end;

procedure TNeuralNetExtended.SaveNetwork;
var
  i, j, k: integer;
begin
  FNnwFile.WriteFloat('Network', 'TeachSpeed', TeachRate);
  FNnwFile.WriteFloat('Network', 'Miu', Momentum);
```

```
  FNnwFile.WriteFloat('Network', 'Alpha', Alpha);
  FNnwFile.WriteInteger('Network', 'Epoch', EpochCount);
  FNnwFile.WriteInteger('Network','CountLayers', LayerCount);
  // задається кількість нейронів у шарах
  for i := 0 to LayerCount - 1 do
    FNnwFile.WriteInteger('Network','Layer_'+IntToStr(i),
StrToInt(NeuronsInLayer[i]));
  // завантаження вагових коефіцієнтів і зсуву
  for i:= 1 to LayerCount - 1 do
    for j := 0 to StrToInt(NeuronsInLayer[i]) - 1 do
    begin
      for k := 0 to StrToInt(NeuronsInLayer[ i-1]) do
        FNnwFile.WriteFloat('Network','W_'+IntToStr( i-1)+'_'+IntToStr(k)+
                            '_'+IntToStr(j),
LayersBP[i].NeuronsBP[j].Weights[k]);
      FNnwFile.WriteFloat('Network','WT_'+IntToStr( i-1)+'_'+IntToStr(j),
         LayersBP[i].NeuronsBP[j].Weights[StrToInt(NeuronsInLayer[j])]);
   end;
end;

procedure TNeuralNetExtended.NormalizeData;
var
  i: integer;
begin
  // нормалізація вхідних і вихідних значень
  for i := 0 to FAvailableFieldsCount - 1 do
  begin
    FFields[i].FindMinMax;
    FFields[i].Normalize;
  end;
end;

procedure TNeuralNetExtended.Train;
var
  i, j, k: integer;
begin
  if FUseForTeach = 100 then
  begin
    PatternCount := FFields[0].DataInCount;
    TestSetPatternCount := 0;
  end
  else
  begin
    PatternCount := Round((FFields[0].DataInCount - 1) * FUseForTeach / 100);
    TestSetPatternCount := FFields[0].DataInCount - PatternCount;
  end;
  if not TeachStopped then
    NormalizeData;
  // формування вхідних значень навчальної множини
  RealOutputIndexCount := OutputFieldCount;
  RealInputIndexCount := InputFieldCount;
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdInput then
    begin
      for j := 0 to PatternCount - 1 do
        FPatternsInput[j, k] := FFields[i].DataIn[j];
      // запам'ятовує індекс поля
      RealInputIndex[k] := i;
      Inc(k);
    end;
  // формування вихідних значень навчальної множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdOutput then
    begin
      for j := 0 to PatternCount - 1 do
        FPatternsOutput[j, k] := FFields[i].DataIn[j];
      // запам'ятовує індекс поля
```

```
      RealOutputIndex[k] := i;
      Inc(k);
    end;
  // формування вхідних значень тестової множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdInput then
    begin
      for j := PatternCount to FFields[i].DataInCount - 1 do
        FTestSetPatterns[j - PatternCount, k] := FFields[i].DataIn[j];
      Inc(k);
    end;
  // формування вихідних значень тестової множини
  k := 0;
  for i := 0 to FAvailableFieldsCount - 1 do
    if FFields[i].KindName = fdOutput then
    begin
      for j := PatternCount to FFields[i].DataInCount - 1 do
        FTestSetPatternsOut[j - PatternCount, k] := FFields[i].DataIn[j];
      Inc(k);
    end;
  // навчання або донавчання мережі
  TeachOffLine;
end;

procedure TNeuralNetExtended.SetAvailableFieldsCount(Value : integer);
var
  i: integer;
begin
  FAvailableFieldsCount := Value;
  // встановлюється кількість полів
  SetLength(FFields, Value);
  for i := 0 to FAvailableFieldsCount - 1 do
    FFields[i] := TNeuroField.Create;
end;

procedure TNeuralNetExtended.SetFields(Index: integer; Value: TNeuroField);
begin
  try
    FFields[Index] := Value;
  except
    on E: ERangeError do
      raise E.CreateFmt(SFieldIndexRange, [Index])
  end;
end;

procedure TNeuralNetExtended.SetDefaultProperties;
begin
  // параметри встановлювані за замовчуванням
  Epoch := False;
  IdentError:= DefaultErrorValue;
  MaxTeachError := False;
  MaxTeachErrorValue := DefaultErrorValue;
  MaxTestError:= False;
  MaxTestErrorValue:= DefaultErrorValue;
  MidTestError:= False;
  MidTestErrorValue:= DefaultErrorValue;
  MidTeachError := False;
  MidTeachErrorValue :=  DefaultErrorValue;
  SettingsLoaded := False;
  TeachIdent := False;
  TeachIdentCount:= DefaultTeachIdentCount;
  TestAsValid := False;
  TestIdent:= False;
  TestIdentCount:= DefaultTestIdentCount;
  UseForTeach := DefaultUseForTeach;
end;

procedure TNeuralNetExtended.SetFileName(Value: TFilename);
```

```
begin
  if Assigned(FNnwFile) then
    FNnwFile.Free;
  try
    FNnwFile := TIniFile.Create(Value);
    FFileName := Value;
  except
    on E: EInOutError do
      raise E.CreateFmt(SWrongFileName, [Value]);
  end;
  FNeuroDataSource := TNeuroDataSource.Create;
  LoadPhase1;
  LoadPhase2;
  LoadPhase4;
  LoadNetwork;
  LoadDataFrom;
end;

procedure TNeuralNetExtended.SetTeachIdentCount(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FTeachIdentCount := DefaultTeachIdentCount
  else
    FTeachIdentCount := Value;
end;

procedure TNeuralNetExtended.SetUseForTeach(const Value: integer);
begin
  if (Value <= 0) or (Value > 100) then
    FUseForTeach := DefaultUseForTeach
  else
    FUseForTeach := Value;
end;

procedure TNeuralNetExtended.SetRealOutputIndex(Index: integer; const Value:
integer);
begin
  FRealOutputIndex[Index] := Value;
end;

procedure TNeuralNetExtended.SetRealOutputIndexCount(const Value: integer);
begin
  SetLength(FRealOutputIndex, Value)
end;
procedure TNeuralNetExtended.SetRealInputIndex(Index: integer; const Value:
integer);
begin
  FRealInputIndex[Index] := Value;
end;
procedure TNeuralNetExtended.SetRealInputIndexCount(const Value: integer);
begin
  SetLength(FRealInputIndex, Value)
end;
procedure Register;
begin
  RegisterComponents('NNTA_', [TNeuralNetHopf, TNeuralNetBP,
TNeuralNetExtended]);
end;
end.
```

Pumpdata.pas – формування бази знань

```
unit PumpData;

interface

uses
  SysUtils, IniFiles, Classes, NNTA_Types;

type

  EFieldNormError = class(Exception);
  EFieldKindError = class(Exception);

  TNeuroField = class;
  TNeuroFields = array of TNeuroField;

  TNeuroField = class(TObject)
  private
    FAlpha: double;
    FDataIn: TVectorFloat;
    FDispersion: double;
    FInd: boolean;
    FKind: byte;
    FName: string;
    FNormType: byte;
    FValueMax: double;
    FValueMid: double;
    FValueMin: double;
    function GetDataIn(Index: integer): double;
    function GetKindName: TNeuroFieldType;
    function GetNormTypeName: TNormalize;
    function GetDataInCount: integer;
    procedure SetDataIn(Index: integer; Value: double);
    procedure SetKind(Value: byte);
    procedure SetNormType(Value: byte);
    procedure SetDataInCount(Value: integer);
  public
    procedure FindMinMax;
    procedure CalcMid;
    procedure CalcDispersion;
    procedure Normalize;
    procedure DeNormalize;
    property Alpha: double read FAlpha write FAlpha;
    property DataIn[Index: integer]: double read GetDataIn write SetDataIn;
    property DataInCount: integer read GetDataInCount write SetDataInCount;
    property Dispersion: double read FDispersion write FDispersion;
    property Ind: boolean read FInd write FInd;
    property Kind: byte read FKind write SetKind;
    property KindName: TNeuroFieldType read GetKindName;
    property Name: string read FName write FName;
    property NormType: byte read FNormType write SetNormType;
    property NormTypeName: TNormalize read GetNormTypeName;
    property ValueMax: double read FValueMax write FValueMax;
    property ValueMin: double read FValueMin write FValueMin;
    property ValueMid: double read FValueMid write FValueMid;
  end;

  TNeuroDataSource = class(TObject)
  private
    FName: TFileName;
    function IsHeaderChar(AValue: char): boolean;
  public
    function FieldCount(AHeader: string): integer;
    procedure ExtractHeaders(const AFields: TNeuroFields; AHeader: string);
    procedure ExtractValues(const AVector:TVectorFloat; AHeader: string);
    property Name: TFileName read FName write FName;
  end;
```

```
implementation

{ Клас TNeuroField }

function TNeuroField.GetDataIn(Index: integer): double;
begin
  Result := FDataIn[Index];
end;

function TNeuroField.GetDataInCount: integer;
begin
  Result := High(FDataIn) + 1;
end;

function TNeuroField.GetKindName: TNeuroFieldType;
begin
  case FKind of
    0 : Result := fdInput;
    1 : Result := fdOutput;
    2 : Result := fdNone;
  end;
end;

function TNeuroField.GetNormTypeName: TNormalize;
begin
  case FNormType of
    0 : if KindName = fdInput then
          Result := nrmLinear
        else if KindName = fdOutput then
          Result := nrmLinearOut;
    1 : Result := nrmSigmoid;
    2 : Result := nrmAuto;
    3 : Result := nrmNone;
  end;
end;

procedure TNeuroField.CalcMid;
var
  i: integer;
begin
  FValueMid := 0;
  for i := Low(FDataIn) to High(FDataIn) do
    FValueMid := FValueMid + FDataIn[i];
  FValueMid := FValueMid/(High(FDataIn) + 1);
end;

procedure TNeuroField.CalcDispersion;
var
  i: integer;
begin
  if High(FDataIn) > 1 then
  begin
    FDispersion := 0;
    for i := Low(FDataIn) to High(FDataIn) do
      FDispersion := FDispersion + sqr(FDataIn[i] - ValueMid);
    FDispersion := sqrt(FDispersion/High(FDataIn));
  end
  else
    FDispersion := 0;
end;

(*procedure TNeuroField.DeNormalize;
var
  i: integer;
  xTmp: double;
begin
  case NormTypeName of
    nrmLinear: for i := Low(FDataIn) to High(FDataIn) do
```

```
                FDataIn[i] := (FDataIn[i] + 1)*(FValueMax - FValueMin)/2 +
FValueMin;
    nrmLinearOut: for i := Low(FDataIn) to High(FDataIn) do
                FDataIn[i] := FDataIn[i]*(FValueMax - FValueMin) + FValueMin;
    nrmSigmoid: for i := Low(FDataIn) to High(FDataIn) do
                FDataIn[i] := - Ln(1/FDataIn[i] - 1)/Alpha;
  end;
end;*)

procedure TNeuroField.FindMinMax;
var
  i: integer;
begin
  FValueMax:= FDataIn[0];
  FValueMin:= FDataIn[0];
  for i := 1 to High(FDataIn) do
  begin
    if FValueMin > FDataIn[i] then
      FValueMin := FDataIn[i];
    if FValueMax < FDataIn[i] then
      FValueMax := FDataIn[i]
  end;
end;

procedure TNeuroField.Normalize;
var
  i: integer;
  xTmp: double;
begin
  case NormTypeName of
    nrmAuto:  begin
                CalcMid;
                CalcDispersion;
                for i := Low(FDataIn) to High(FDataIn) do
                begin
                  xTmp := (FDataIn[i] - FValueMid)/FDispersion;
                  FDataIn[i] := 1/(1 + exp(-xTmp));
                end;
              end;
    nrmLinear: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := 2*(FDataIn[i] - FValueMin)/(FValueMax - FValueMin)-
1;
    nrmLinearOut: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := (FDataIn[i] - FValueMin)/(FValueMax - FValueMin);
    nrmSigmoid: for i := Low(FDataIn) to High(FDataIn) do
               FDataIn[i] := 1/( 1 + exp(-Alpha * FDataIn[i]));
  end;
end;

procedure TNeuroField.SetNormType(Value: byte);
begin
  if (Value < 0) or (Value > 3) then
    raise EFieldNormError.CreateFmt(SFieldNorm, [Value])
  else
    FNormType := Value;
end;

procedure TNeuroField.SetKind(Value: byte);
begin
  if (Value < 0) or (Value > 2) then
    raise Exception.CreateFmt(SFieldKind, [Value])
  else
    FKind := Value;
end;

procedure TNeuroField.SetDataIn(Index: integer; Value: double);
begin
  FDataIn[Index] := Value;
end;
```

```
procedure TNeuroField.SetDataInCount(Value: integer);
begin
  SetLength(FDataIn, Value)
end;

{ Клас TNeuroDataSource }

function TNeuroDataSource.IsHeaderChar(AValue: char): boolean;
begin
  if (AValue in Letters) or (AValue in Capitals) or (AValue in DigitChars) then
    Result := True
  else
    Result := False;
end;

procedure TNeuroDataSource.ExtractValues(const AVector:TVectorFloat; AHeader:
string);
var
  s: string;
  i, xCurPos: integer;
begin
  i := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  try
    while xCurPos > 0 do
    begin
      s := Copy(AHeader, 1, xCurPos - 1);
      AVector[i] := StrToFloat(s);
      Inc(i);
      Delete(AHeader, 1, xCurPos - 1);
      AHeader := Trim(AHeader);
      xCurPos := Pos(SpaceChar, AHeader);
    end;
    s := AHeader;
    AVector[i] := StrToFloat(s);
  except
    on EConvertError do
      EConvertError.CreateFmt(SCannotBeNumber, [s])
  end;
end;

procedure TNeuroDataSource.ExtractHeaders(const AFields: TNeuroFields; AHeader:
string);
var
  s: string;
  xFieldCount, j, xCurPos: integer;
begin
  { виділяє заголовки з файлу }
  xFieldCount := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  while xCurPos > 0 do
  begin
    s := Copy(AHeader, 1, xCurPos - 1);
    AFields[xFieldCount].FName := '';
    for j := 1 to Length(s) do
      if isHeaderChar(s[j]) then
        AFields[xFieldCount].FName := AFields[xFieldCount].FName + s[j];
    Inc(xFieldCount);
    Delete(AHeader, 1, xCurPos - 1);
    AHeader := Trim(AHeader);
    xCurPos := Pos(SpaceChar, AHeader);
  end;
  AFields[xFieldCount].FName := '';
  for j := 1 to Length(AHeader) do
    if isHeaderChar(AHeader[j]) then
      AFields[xFieldCount].FName := AFields[xFieldCount].FName + AHeader[j];
```

```
  { повертає кількість полів }
end;

function TNeuroDataSource.FieldCount(AHeader: string): integer;
var
  xFieldCount, xCurPos: integer;
begin
  { виділяє заголовки з файлу }
  xFieldCount := 0;
  AHeader := Trim(AHeader);
  xCurPos := Pos(SpaceChar, AHeader);
  while xCurPos > 0 do
  begin
    Inc(xFieldCount);
    Delete(AHeader, 1, xCurPos - 1);
    AHeader := Trim(AHeader);
    xCurPos := Pos(SpaceChar, AHeader);
  end;
  { повертає кількість полів }
  Result := xFieldCount + 1;
end;


end.
```

NeuralNetExtend.pas – навчання мережі

```
unit NeuralNetExtend;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  ComCtrls, NNTA_Comp, ExtCtrls, StdCtrls, Spin, Grids, NNTA_Types,
  IniFiles;

const
  FormCaption = 'Навчання мережі';

type
  TfrmNeuralNetExtend = class(TForm)
    PageControl: TPageControl;
    pnlNavigation: TPanel;
    Tab1: TTabSheet;
    btnBack: TButton;
    rgrFileType: TRadioGroup;
    btnNext: TButton;
    btnCancel: TButton;
    Tab2: TTabSheet;
    lblFileName: TLabel;
    btnOpenFile: TButton;
    edtFileName: TEdit;
    OpenDialog: TOpenDialog;
    Tab3: TTabSheet;
    ltbFieldName: TListBox;
    Label2: TLabel;
    rdgFieldType: TRadioGroup;
    rdgNormType: TRadioGroup;
    GroupBox1: TGroupBox;
    Label3: TLabel;
    edtMin: TEdit;
    Label4: TLabel;
    edtMax: TEdit;
    Label5: TLabel;
    edt: TEdit;
    Tab4: TTabSheet;
    speLayers: TSpinEdit;
    Label6: TLabel;
    stgNeuronsInLayer: TStringGrid;
    Label7: TLabel;
    Tab5: TTabSheet;
    Label8: TLabel;
    tbrAlpha: TTrackBar;
    sttAlpha: TStaticText;
    Label9: TLabel;
    edtMomentum: TEdit;
    Label10: TLabel;
    edtTeachRate: TEdit;
    Tab6: TTabSheet;
    Label11: TLabel;
    Label12: TLabel;
    btnContinueTeach: TButton;
    sttMaxTeachError: TStaticText;
    sttEpochCount: TStaticText;
    GroupBox2: TGroupBox;
    Label13: TLabel;
    edtIdentError: TEdit;
    speEpochCount: TSpinEdit;
    cbxEpoch: TCheckBox;
    cbxMaxTeachError: TCheckBox;
    edtMaxTeachErrorValue: TEdit;
    cbxMidTeachError: TCheckBox;
```

```
    edtMidTeachErrorValue: TEdit;
    cbxTeachIdent: TCheckBox;
    speTeachIdentValue: TSpinEdit;
    btnBeginTeach: TButton;
    cbxMaxTestError: TCheckBox;
    cbxMidTestError: TCheckBox;
    cbxTestIdent: TCheckBox;
    edtMaxTestErrorValue: TEdit;
    edtMidTestErrorValue: TEdit;
    speTestIdentValue: TSpinEdit;
    Tab7: TTabSheet;
    Label14: TLabel;
    stgInput: TStringGrid;
    Label15: TLabel;
    stgOutput: TStringGrid;
    btnCompute: TButton;
    Memo1: TMemo;
    Label16: TLabel;
    Label17: TLabel;
    Memo2: TMemo;
    Bevel1: TBevel;
    Memo3: TMemo;
    Label1: TLabel;
    sttMaxTestError: TStaticText;
    Label18: TLabel;
    sttMidTeachError: TStaticText;
    Label19: TLabel;
    sttMidTestError: TStaticText;
    SaveDialog: TSaveDialog;
    edtUseForTeach: TEdit;
    Label20: TLabel;
    btnSave: TButton;
    procedure btnCancelClick(Sender: TObject);
    procedure btnNextClick(Sender: TObject);
    procedure btnBackClick(Sender: TObject);
    procedure btnOpenFileClick(Sender: TObject);
    procedure FormActivate(Sender: TObject);
    procedure ltbFieldNameClick(Sender: TObject);
    procedure rdgFieldTypeClick(Sender: TObject);
    procedure rdgNormTypeClick(Sender: TObject);
    procedure edtAChange(Sender: TObject);
    procedure tbrAlphaChange(Sender: TObject);
    procedure edtMomentumChange(Sender: TObject);
    procedure edtTeachRateChange(Sender: TObject);
    procedure NeuralNetExtendedEpochPassed(Sender: TObject);
    procedure btnContinueTeachClick(Sender: TObject);
    procedure edtIdentErrorChange(Sender: TObject);
    procedure cbxEpochClick(Sender: TObject);
    procedure speEpochCountChange(Sender: TObject);
    procedure cbxMaxTeachErrorClick(Sender: TObject);
    procedure edtMaxTeachErrorValueChange(Sender: TObject);
    procedure cbxMidTeachErrorClick(Sender: TObject);
    procedure edtMidTeachErrorValueChange(Sender: TObject);
    procedure cbxTeachIdentClick(Sender: TObject);
    procedure speTeachIdentValueChange(Sender: TObject);
    procedure cbxMaxTestErrorClick(Sender: TObject);
    procedure edtMaxTestErrorValueChange(Sender: TObject);
    procedure cbxMidTestErrorClick(Sender: TObject);
    procedure edtMidTestErrorValueChange(Sender: TObject);
    procedure cbxTestIdentClick(Sender: TObject);
    procedure speTestIdentValueChange(Sender: TObject);
    procedure NeuralNetExtendedAfterTeach(Sender: TObject);
    procedure btnBeginTeachClick(Sender: TObject);
    procedure btnComputeClick(Sender: TObject);
    procedure speLayersChange(Sender: TObject);
    procedure btnSaveClick(Sender: TObject);
    procedure edtUseForTeachChange(Sender: TObject);
  private
    { Private declarations }
```

```
    Teach: boolean;
    NotSaved: boolean;
    procedure ChangePage;
    procedure OpenFile;
    procedure LoadFile;
    procedure Tune;
    procedure CreateNet;
    procedure RunTeach;
    procedure TestNet;
  public
    { Public declarations }
  end;

var
  frmNeuralNetExtend: TfrmNeuralNetExtend;
implementation

{$R *.DFM}

// Запуск програми
procedure TfrmNeuralNetExtend.FormActivate(Sender: TObject);
begin
  Caption := FormCaption;
  PageControl.ActivePage := PageControl.Pages[0];
  Teach := false;
  NotSaved := false;
end;

// Вихід із програми
procedure TfrmNeuralNetExtend.btnCancelClick(Sender: TObject);
begin
  if NotSaved then
    if MessageDlg('Є незбережені дані. Зберегти?',mtConfirmation, [mbYes, mbNo],
0) = mrYes then
      btnSave.Click;
  Close;
end;

// Натискання кнопки "Вперед"
procedure TfrmNeuralNetExtend.btnNextClick(Sender: TObject);
begin
  with PageControl do
  begin
    ActivePage := FindNextPage(ActivePage, true, false);
    ChangePage;
    if ActivePage.PageIndex = PageCount - 1 then
      btnNext.Enabled := false
    else
      btnNext.Enabled := true;
    btnBack.Enabled := true;
  end;
end;

// Натискання кнопки "Назад"
procedure TfrmNeuralNetExtend.btnBackClick(Sender: TObject);
begin
  with PageControl do
  begin
    ActivePage := FindNextPage(ActivePage, false, false);
    if ActivePage.PageIndex = 0 then
      btnBack.Enabled := false
    else
      btnBack.Enabled := true;
    btnNext.Enabled := true;
  end;
end;

// Дія на зміну сторінки
procedure TfrmNeuralNetExtend.ChangePage;
```

```
begin
  case PageControl.ActivePage.PageIndex of
    1: OpenFile;
    2: LoadFile;
    3: Tune;
    4: CreateNet;
    6: TestNet;
  end;
end;

// Вибрати файл - джерело даних
procedure TfrmNeuralNetExtend.OpenFile;
begin
  if rgrFileType.ItemIndex = 0 then
  begin
    OpenDialog.Filter := 'NNW files (*.nnw)|*.nnw';
    lblFileName.Caption := 'Виберіть nnw-файл';
  end
  else
  begin
    OpenDialog.Filter := 'Text files (*.txt)|*.txt';
    lblFileName.Caption := 'Виберіть txt-файл';
  end;
end;

// Вибір файлу в діалоговому вікні
procedure TfrmNeuralNetExtend.btnOpenFileClick(Sender: TObject);
begin
  OpenDialog.Execute;
  Caption := FormCaption + ' - ' + ExtractFileName(OpenDialog.FileName);
  edtFileName.Text := OpenDialog.FileName;
end;

// Завантажити в компонент обраний файл
procedure TfrmNeuralNetExtend.LoadFile;
var
  i: integer;
begin
  try
    if rgrFileType.ItemIndex = 0 then
      NeuralNetExtended.FileName := edtFileName.Text   // nnw-файл
    else
    begin
      NeuralNetExtended.SourceFileName := edtFileName.Text; // текстовий файл
      NeuralNetExtended.LoadDataFrom; // завантажує дані з текстового файлу
      // конфігурація нейронної мережі за замовчуванням
      NeuralNetExtended.AddLayer(2);
      NeuralNetExtended.AddLayer(3);
      NeuralNetExtended.AddLayer(1);
    end;
  except
    raise Exception.Create('Помилка при відкритті файлу');
  end;
  NeuralNetExtended.Init;  // Ініціалізація мережі
  // Формування списку полів для StringList-A
  ltbFieldName.Clear;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    ltbFieldName.Items.Add(NeuralNetExtended.Fields[i].Name);
  ltbFieldName.ItemIndex := 0;
  ltbFieldNameClick(Self);
end;

// Настроювання параметрів мережі
procedure TfrmNeuralNetExtend.Tune;
var
  i: integer;
begin
  speLayers.Value := NeuralNetExtended.LayerCount - 2;
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
```

```
  stgNeuronsInLayer.Cells[0, 0] := '№ шаруючи';
  stgNeuronsInLayer.Cells[1, 0] := 'Нейронів';
  for i := 0 to speLayers.Value - 1 do
  begin
    stgNeuronsInLayer.Cells[0, i + 1] := IntToStr(i);
    stgNeuronsInLayer.Cells[1, i + 1] := IntToStr(NeuralNetExtended.Layers[i +
1].NeuronCount);
  end;

  tbrAlpha.Position := trunc(NeuralNetExtended.Alpha * 100);
  edtMomentum.Text := FloatToStr(NeuralNetExtended.Momentum);
  edtTeachRate.Text := FloatToStr(NeuralNetExtended.TeachRate);
  edtIdentError.Text := FloatToStr(NeuralNetExtended.IdentError);
  edtUseForTeach.Text := FloatToStr(NeuralNetExtended.UseForTeach);

  cbxEpoch.Checked := NeuralNetExtended.Epoch;
  speEpochCount.Text := IntToStr(NeuralNetExtended.EpochCount);

  cbxMaxTeachError.Checked := NeuralNetExtended.MaxTeachError;
  edtMaxTeachErrorValue.Text :=
FloatToStr(NeuralNetExtended.MaxTeachErrorValue);

  cbxMidTeachError.Checked := NeuralNetExtended.MidTeachError;
  edtMidTeachErrorValue.Text :=
FloatToStr(NeuralNetExtended.MidTeachErrorValue);

  cbxTeachIdent.Checked := NeuralNetExtended.TeachIdent;
  speTeachIdentValue.Value := NeuralNetExtended.TeachIdentCount;

  cbxMaxTestError.Checked := NeuralNetExtended.MaxTestError;
  edtMaxTestErrorValue.Text := FloatToStr(NeuralNetExtended.MaxTestErrorValue);

  cbxMidTestError.Checked := NeuralNetExtended.MidTestError;
  edtMidTestErrorValue.Text := FloatToStr(NeuralNetExtended.MidTestErrorValue);

  cbxTestIdent.Checked := NeuralNetExtended.TestIdent;
  speTestIdentValue.Value := NeuralNetExtended.TeachIdentCount;
end;

// Відображення інформації про обране поле (тип поля, нормалізація та інше)
procedure TfrmNeuralNetExtend.ltbFieldNameClick(Sender: TObject);
begin
  // Тип поля - вхідне, вихідне, не використовувати
  rdgFieldType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind;
  // Тип нормалізації
  rdgNormType.ItemIndex :=
NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType;
  // Параметр нормалізації
  edt.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha);
  // Мінінум та максимум (для лінійної нормалізації)
  edtMin.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMin);
  edtMax.Text :=
FloatToStr(NeuralNetExtended.Fields[ltbFieldName.ItemIndex].ValueMax);
end;

// Змінити тип поля
procedure TfrmNeuralNetExtend.rdgFieldTypeClick(Sender: TObject);
begin
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Kind :=
rdgFieldType.ItemIndex
end;

// Змінити тип нормалізації
procedure TfrmNeuralNetExtend.rdgNormTypeClick(Sender: TObject);
begin
```

```
  NeuralNetExtended.Fields[ltbFieldName.ItemIndex].NormType :=
rdgNormType.ItemIndex
end;

// Змінити параметр нормалізації
procedure TfrmNeuralNetExtend.edtAChange(Sender: TObject);
begin
 NeuralNetExtended.Fields[ltbFieldName.ItemIndex].Alpha := StrToFloat(edt.Text);
end;

// Змінити параметр Alpha мережі
procedure TfrmNeuralNetExtend.tbrAlphaChange(Sender: TObject);
begin
  sttAlpha.Caption := FloatToStr(tbrAlpha.Position / 100);
end;

// Зміна кількості схованих шарів
procedure TfrmNeuralNetExtend.speLayersChange(Sender: TObject);
begin
  stgNeuronsInLayer.RowCount := speLayers.Value + 1;
end;

// Створення мережі обраної топології
procedure TfrmNeuralNetExtend.CreateNet;
var
  i: integer;
  xInput, xOutput: integer;
begin
  // Змінюється тільки кількість нейронів у схованих шарах,
  // Кількість нейронів у вхідному й вихідному шарі залежить від
  // типів полів
  with NeuralNetExtended do
  begin
    xInput := InputFieldCount;
    xOutput := OutputFieldCount;
    ResetLayers;
    AddLayer(xInput);
    for i := 0 to speLayers.Value - 1 do
      AddLayer(StrToInt(stgNeuronsInLayer.Cells[1, i + 1]));
    AddLayer(xOutput);
  end;
end;

// Змінити момент мережі
procedure TfrmNeuralNetExtend.edtMomentumChange(Sender: TObject);
begin
  NeuralNetExtended.Momentum := StrToFloat(edtMomentum.Text);
end;

// Змінити швидкість навчання мережі
procedure TfrmNeuralNetExtend.edtTeachRateChange(Sender: TObject);
begin
  NeuralNetExtended.TeachRate := StrToFloat(edtTeachRate.Text);
end;

// Дія по проходженню однієї епохи навчання
procedure TfrmNeuralNetExtend.NeuralNetExtendedEpochPassed(Sender: TObject);
begin
  sttMaxTestError.Caption := '';
  sttMidTestError.Caption := '';
  with NeuralNetExtended do
  begin
    sttMaxTeachError.Caption := FloatToStr(MaxTeachResidual); // Показати макс.
помилку на навчальній множині
    sttMidTeachError.Caption := FloatToStr(MidTeachResidual); // Показати серед.
помилку на навчальній множині
    if NeuralNetExtended.UseForTeach <> 100 then
    begin
```

```
      sttMaxTestError.Caption := FloatToStr(MaxTestResidual); // Показати макс.
помилку на тестовій множині
      sttMidTestError.Caption := FloatToStr(MidTestResidual); // Показати сред.
помилку на тестовій множині
    end;
    sttEpochCount.Caption := FloatToStr(EpochCurrent); // Показати номер
поточної епохи
  end;
  Application.ProcessMessages; // Дати можливість Windows перемалювати форму
end;

// Натискання кнопки "Продовжити навчання"
procedure TfrmNeuralNetExtend.btnContinueTeachClick(Sender: TObject);
begin
  NeuralNetExtended.ContinueTeach := true; // Скинути прапор - "Продовжити
навчання"
  RunTeach;  // Запуск
end;

// Натискання кнопки "Навчити"
procedure TfrmNeuralNetExtend.btnBeginTeachClick(Sender: TObject);
begin
  NeuralNetExtended.ContinueTeach := false; // Скинути прапор - "Почати навчання
знову"
  RunTeach;
end;

procedure TfrmNeuralNetExtend.edtIdentErrorChange(Sender: TObject);
begin
  NeuralNetExtended.IdentError := StrToFloat(edtIdentError.Text);
end;

procedure TfrmNeuralNetExtend.edtUseForTeachChange(Sender: TObject);
begin
  NeuralNetExtended.UseForTeach := StrToInt(edtUseForTeach.Text);
end;

procedure TfrmNeuralNetExtend.cbxEpochClick(Sender: TObject);
begin
  NeuralNetExtended.Epoch := cbxEpoch.Checked;
end;

procedure TfrmNeuralNetExtend.speEpochCountChange(Sender: TObject);
begin
  NeuralNetExtended.EpochCount := StrToInt(speEpochCount.Text);
end;

procedure TfrmNeuralNetExtend.cbxMaxTeachErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MaxTeachError := cbxMaxTeachError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMaxTeachErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MaxTeachErrorValue :=
StrToFloat(edtMaxTeachErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxMidTeachErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MidTeachError := cbxMidTeachError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMidTeachErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MidTeachErrorValue :=
StrToFloat(edtMidTeachErrorValue.Text);
end;
```

```
procedure TfrmNeuralNetExtend.cbxTeachIdentClick(Sender: TObject);
begin
  NeuralNetExtended.TeachIdent := cbxTeachIdent.Checked;
end;

procedure TfrmNeuralNetExtend.speTeachIdentValueChange(Sender: TObject);
begin
  NeuralNetExtended.TeachIdentCount := speTeachIdentValue.Value;
end;

procedure TfrmNeuralNetExtend.cbxMaxTestErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MaxTestError := cbxMaxTestError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMaxTestErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MaxTestErrorValue := StrToFloat(edtMaxTestErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxMidTestErrorClick(Sender: TObject);
begin
  NeuralNetExtended.MidTestError := cbxMidTestError.Checked;
end;

procedure TfrmNeuralNetExtend.edtMidTestErrorValueChange(Sender: TObject);
begin
  NeuralNetExtended.MidTestErrorValue := StrToFloat(edtMidTestErrorValue.Text);
end;

procedure TfrmNeuralNetExtend.cbxTestIdentClick(Sender: TObject);
begin
  NeuralNetExtended.TestIdent := cbxTestIdent.Checked;
end;

procedure TfrmNeuralNetExtend.speTestIdentValueChange(Sender: TObject);
begin
  NeuralNetExtended.TeachIdentCount := speTestIdentValue.Value;
end;

// Зупинка навчання
procedure TfrmNeuralNetExtend.NeuralNetExtendedAfterTeach(Sender: TObject);
begin
  btnBack.Enabled := true;
  btnNext.Enabled := true;
  btnCancel.Enabled := true;
  btnContinueTeach.Caption := 'Навчити';
  NeuralNetExtended.StopTeach := true;
end;

procedure TfrmNeuralNetExtend.RunTeach;
begin
  Teach := not Teach;  // Перемикач стану "вчимося/не вчимося"
  if Teach then
  begin
    btnBeginTeach.Enabled := false;
    btnBack.Enabled := false;
    btnNext.Enabled := false;
    btnCancel.Enabled := false;
    NeuralNetExtended.StopTeach := false;
    btnContinueTeach.Caption := 'Зупинити навчання';
    NotSaved := true;
    NeuralNetExtended.Train;  // Запуск нейромережі на навчання
    btnCompute.Enabled := true;
    btnSave.Enabled := true;
  end
  else
  begin
    btnBeginTeach.Enabled := true;
```

```
    btnBack.Enabled := true;
    btnNext.Enabled := true;
    btnCancel.Enabled := true;
    btnContinueTeach.Caption := 'Продовжити навчання';
    NeuralNetExtended.StopTeach := true; // Зупинити навчання
  end;
end;

// Відкриття сторінки - тестування навченої нейромережі
procedure TfrmNeuralNetExtend.TestNet;
var
  i, j: integer;
begin
  stgInput.RowCount := NeuralNetExtended.InputFieldCount + 1;
  stgInput.Cells[0, 0] := 'Поле';
  stgInput.Cells[1, 0] := 'Значення';

  // Проставити імена вхідних полів
  j := 0;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    if (NeuralNetExtended.Fields[i].KindName = fdInput) then // Ознака того, що
поле вхідне
    begin
      Inc(j);
      stgInput.Cells[0, j] := NeuralNetExtended.Fields[i].Name;
    end;
  stgOutput.RowCount := NeuralNetExtended.OutputFieldCount + 1;
  stgOutput.Cells[0, 0] := 'Поле';
  stgOutput.Cells[1, 0] := 'Значення';

  // Проставити імена вихідних полів
  j := 0;
  for i := 0 to NeuralNetExtended.AvailableFieldsCount - 1 do
    if (NeuralNetExtended.Fields[i].KindName = fdOutput) then  // Ознака того,
що поле вихідне
    begin
      Inc(j);
      stgOutput.Cells[0, j] := NeuralNetExtended.Fields[i].Name;
    end;
end;

// Натискання кнопки "Обчислити"
procedure TfrmNeuralNetExtend.btnComputeClick(Sender: TObject);
var
  xVectorFloat: TVectorFloat;
  i: integer;
begin
  // Створити вектор, що будемо подавати на вхід
  // довжиною, рівною кількості нейронів на вхідному шарі
  SetLength(xVectorFloat, NeuralNetExtended.InputFieldCount);
  // Заповнити значення елементів вектора
  for i := 0 to NeuralNetExtended.InputFieldCount - 1 do
    xVectorFloat[i] := StrToFloat(stgInput.Cells[1, i + 1]);
  // Подати на вхід нейромережі. Результати будуть у вихідному шарі нейромережі
  NeuralNetExtended.ComputeUnPrepData(xVectorFloat);
  // Відобразити отримані результати
  for i := 0 to NeuralNetExtended.OutputFieldCount - 1 do
    stgOutput.Cells[1, i + 1] := FloatToStr(NeuralNetExtended.Output[i]);
  // Знищити вектор
  SetLength(xVectorFloat, 0);
  xVectorFloat := nil;
end;

// Зберегти навчену нейромережу
procedure TfrmNeuralNetExtend.btnSaveClick(Sender: TObject);
begin
  SaveDialog.InitialDir := ExtractFilePath(NeuralNetExtended.FileName);
  SaveDialog.FileName := ExtractFileName(NeuralNetExtended.FileName);
  if SaveDialog.Execute then
```

```
  begin
    NeuralNetExtended.NnwFile := TIniFile.Create(SaveDialog.FileName);
    NeuralNetExtended.SavePhase1;
    NeuralNetExtended.SavePhase2;
    NeuralNetExtended.SavePhase4;
    NeuralNetExtended.SaveNetwork;
    NotSaved := false;
  end;
end;

end.
```

Hopf.pas - Блок формування та розпізнавання образів

```
unit Hopf;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,
  Grids, NNTA_Comp, NNTA_Types, Db, DBTables, ExtCtrls, DBCtrls, StdCtrls,
  ToolWin, ComCtrls;

type
  TForm1 = class(TForm)
    Table: TTable;
    btnExecute: TButton;
    DBNavigator: TDBNavigator;
    DataSource: TDataSource;
    btnEdit: TButton;
    stgDatabase: TStringGrid;
    stgInput: TStringGrid;
    stgOutput: TStringGrid;
    NeuralNetHopf: TNeuralNetHopf;
    StaticText1: TStaticText;
    StaticText2: TStaticText;
    StaticText3: TStaticText;
    Bevel: TBevel;
    TableLETTERS: TStringField;
    dbMain: TDatabase;
    procedure DataSourceDataChange(Sender: TObject; Field: TField);
    procedure FormActivate(Sender: TObject);
    procedure GridClick(Sender: TObject);
    procedure btnEditClick(Sender: TObject);
    procedure btnExecuteClick(Sender: TObject);
    procedure GridDrawCell(Sender: TObject; ACol, ARow: Integer;
      Rect: TRect; State: TGridDrawState);
    procedure FormCreate(Sender: TObject);
  public
    { Public declarations }
    procedure AddPattern(Value: string);
    procedure Clear(Grid: TStringGrid);
    procedure Init;
    procedure ShowMatrix(Grid: TStringGrid; Value: string);
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

// Показати вектор у вигляді сітки
procedure TForm1.ShowMatrix(Grid: TStringGrid; Value: string);
var
  i, j: integer;
begin
  Clear(Grid);
  for i := 0 to Grid.ColCount - 1 do
    for j := 0 to Grid.RowCount - 1 do
    begin
      try
        if Value[i * Grid.RowCount + j + 1] = '1' then
          Grid.Cells[i, j] := '1'
        else
          Grid.Cells[i, j] := ' '
      except
        Grid.Cells[i, j] := ' '
      end;
```

```
    end;
end;

// Очистити сітку
procedure TForm1.Clear(Grid: TStringGrid);
var
  i, j: integer;
begin
  for i := 0 to Grid.ColCount - 1 do
    for j := 0 to Grid.RowCount - 1 do
      Grid.Cells[i, j] := ' ';
end;

// Показати символ з таблиці
procedure TForm1.DataSourceDataChange(Sender: TObject; Field: TField);
begin
  ShowMatrix(stgDatabase, TableLETTERS.Value);
end;

// Ініціалізація мережі значеннями з таблиці
procedure TForm1.Init;
begin
  // Очистити мережу від зразків
  NeuralNetHopf.ResetPatterns;

  // Додати зразки з таблиці до мережі
  Table.First;
  while not Table.Eof do
  begin
    AddPattern(TableLETTERS.AsString);
    Table.Next;
  end;
  Table.First;

  // Ініціалізувати ваги
  NeuralNetHopf.InitWeights;
  // Мережа підготовлена до розпізнавання
end;

procedure TForm1.FormActivate(Sender: TObject);
begin
  Clear(stgDatabase);
  Clear(stgInput);
  Clear(stgOutput);
  Init;
end;

procedure TForm1.GridClick(Sender: TObject);
begin
  with Sender as TStringGrid do
    if Cells[Col, Row] = '1' then
      Cells[Col, Row] := ' '
    else
      Cells[Col, Row] := '1'
end;

// Додавання нового образа до мережі
procedure TForm1.AddPattern(Value: string);
var
  i: integer;
  xVector: TVectorInt;
begin
  SetLength(xVector, stgDatabase.RowCount * stgDatabase.ColCount);

  // Перетворення символьного рядка у вектор
  for i := 1 to stgDatabase.RowCount * stgDatabase.ColCount do
    try
      if TableLETTERS.AsString[i] = '1' then
        xVector[i - 1] := 1
```

```
      else
        xVector[i - 1] := -1;
    except
      xVector[i - 1] := -1;
    end;


  NeuralNetHopf.AddPattern(xVector);
end;

procedure TForm1.btnEditClick(Sender: TObject);
var
  i, j: integer;
  xString: string;
begin
  xString := '';
  for i := 0 to stgDatabase.ColCount - 1 do
    for j := 0 to stgDatabase.RowCount - 1 do
      if stgDatabase.Cells[i, j] = '1' then
        xString := xString + '1'
      else
        xString := xString + ' ';
  Table.Edit;
  TableLETTERS.AsString := xString;
  Table.Post;
end;

// Розпізнавання символу
procedure TForm1.btnExecuteClick(Sender: TObject);
var
  i, j: integer;
  xString: string;
begin
  // Подаємо сигнали на вихід мережі
  for i := 0 to stgInput.ColCount - 1 do
    for j := 0 to stgInput.RowCount - 1 do
      if stgInput.Cells[i, j] = '1' then
        NeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount + j].Output := 1
      else
        NeuralNetHopf.Layers[1].Neurons[i * stgInput.RowCount + j].Output := -1;

  // Запуск процесу розпізнавання
  NeuralNetHopf.Calc;

  // Перетворення виходів мережі до рядка
  xString := '';
  for i := 1 to stgOutput.RowCount * stgOutput.ColCount do
    if NeuralNetHopf.Layers[1].Neurons[i - 1].Output = 1 then
      xString := xString + '1'
    else
      xString := xString + ' ';

  // Відобразити результат
  ShowMatrix(stgOutput, xString);
end;

procedure TForm1.GridDrawCell(Sender: TObject; ACol, ARow: Integer;
  Rect: TRect; State: TGridDrawState);
begin
  with Sender as TStringGrid do
  begin
    Canvas.Brush.Color := clBlack;
    if Cells[ACol,ARow] <> ' ' then
      Canvas.FillRect(Rect)
  end;
end;

procedure TForm1.FormCreate(Sender: TObject);
begin
```

```
  dbMain.Params.Values['Path'] := ExtractFilePath(Application.ExeName);
  dbMain.Open;
  Table.Open;

end;

end.
```