МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Навчально-науковий інститут новітніх освітніх технологій
Кафедра спеціалізованих комп'ютерних систем

КОГУТ Юлія Володимирівна

СИСТЕМА АВТОМАТИЗОВАНОГО РОЗПІЗНАВАННЯ ГОЛОСОВОЇ ІНФОРМАЦІЇ / SYSTEM OF AUTOMATED RECOGNITION OF VOICE INFORMATION

спеціальність: 151 – Автоматизація та комп'ютерно-інтегровані технології
освітньо-професійна програма – Автоматизація та комп'ютерно-інтегровані технології

Кваліфікаційна робота

Виконав студент групи АКІТзм-21
Ю. В. Когут (Дрожевська)
_____
Науковий керівник:
к.т.н., доцент А. І. Сегін

Випускну кваліфікаційну роботу
допущено до захисту:
"____"_____20___ р.

Завідувач кафедри СКС
_____ **А. І. Сегін**

ТЕРНОПІЛЬ – 2023

Факультет комп'ютерних інформаційних технологій
Кафедра спеціалізованих комп'ютерних систем
Освітній ступінь "магістр"
спеціальність:151 – Автоматизація та комп'ютерно-інтегровані технології
освітньо-професійна програма – Автоматизація та комп'ютерно-інтегровані технології

ЗАТВЕРДЖУЮ:
зав. кафедри СКС
_____ А. І. Сегін
26 жовтня 2022р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
КОГУТ Юлії Володимирівні
(прізвище, ім'я по-батькові)

1. Тема кваліфікаційної роботи
Система автоматизованого розпізнавання голосової інформації / System of automated recognition of voice information.
керівник роботи к.т.н., доцент Сегін А. І.

затверджено наказом по університету від «8» грудня 2022 р. № 491

2. Строк подання студентом закінченої кваліфікаційної роботи
30 листопала 2023р.

3. Вихідні дані до кваліфікаційної роботи:
1. Основні трудності у вирішенні задачі РМАП та на основі сучасного досвіду.
2. Сучасне технічне обладнання для цифрового запису та обробки аудіо сигналів.
3. Вимоги до системи розпізнавання мовленнєвих аудіо потоків.
4. Сучасний стан розвитку теорії нейронних мереж.

4. Основні питання, які потрібно розробити
1. Проаналізувати сучасний стан розвитку систем автоматичного розпізнавання мовленнєвих аудіо потоків (РМАП).
2. Проаналізувати структури і функціонування нейронних мереж та обґрунтувати обрання структур нейромереж.
3. Розробити структурну та функціональну схему системи РМАП.
4. Розробити структурну схему та алгоритм роботи програми розпізнавання мовленнєвих аудіо потоків.
5. Провести моделювання роботи РМАП. та розробити програмне забезпечення для реалізації методу РМАП.

5. Перелік графічного матеріалу у роботі
1. Загальна структурна схема системи розпізнавання мовленнєвих аудіо

потоків.
2. Архітектура нейромережі для розпізнавання мовлення.
3. Структурна схема програми розпізнавання мовленнєвих аудіо потоків.
4. Блок-схема роботи програми розпізнавання мовленнєвих аудіо потоків

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1	Сегін А. І., зав. кафедри СКС		
2	Сегін А. І., зав. кафедри СКС		
3	Сегін А. І., зав. кафедри СКС		

7. Дата видачі завдання 20 жовтня 2022р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назви етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз основних проблем автоматичного розпізнавання мовлення та огляд існуючих систем	20.10.2022р. – 28.02.2023р.	виконано
2	Реалізація методу та структури системи розпізнавання мовленнєвих аудіо потоків на основі нейромереж	1.03.2023р. – 30.06.2023р.	виконано
3	Реалізація автоматичної системи розпізнавання мовленнєвих аудіо потоків на основі нейромережі	1.07.2023р. – 5.11.2023р.	виконано
4	Остаточне оформлення та подача кваліфікаційної роботи на перевірку щодо плагіату	6.11.2023р. – 30.11.2023р	виконано

Студент _____ (підпис) Когут Ю. В.

Керівник роботи _____ (підпис) Сегін А. І.

РЕФЕРАТ

Робота виконана на 75 сторінках та містить 23 рисунків, 1 таблицю, 41 джерело за переліком посилань.

Мета кваліфікаційної роботи є розроблення системи автоматичного розпізнавання мовленнєвої аудіо інформації із застосуванням нейронних мереж.

Результати роботи. В результаті досліджень розроблена система автоматичного розпізнавання мовленнєвих аудіо потоків дозволяє правильно перевести мовлення у текст, який набагато простіше опрацьовувати машинним способом.

Рекомендації по використанню результатів роботи. Вирішення задачі достовірного розпізнавання мовленнєвих аудіо потоків відкриває широкі перспективи для розроблення систем автоматичного перекладу, систем автоматизованого управління голосом технічними пристроями, семантичного аналізу голосових аудіо потоків та багато інших сфер застосування.

Ключові слова: РОЗПІЗНАВАННЯ МОВЛЕННЄВИХ АУДІО СИГНАЛІВ, НЕЙРОННА МЕРЕЖА, НЕЙРОННА МЕРЕЖА КОХОНЕНА, НЕЙРОННА МЕРЕЖА ГРОССБЕРГА.

ABSTRACT

The work is completed on 85 pages and contains 23 figures, 1 tables, 41 sources according to the list of references.

The purpose of the qualification is to develop a system for automatic recognition of speech audio information using neural networks.

Work results. As a result of research, a system of automatic recognition of speech audio streams was developed, which allows you to correctly translate speech into text, which is much easier to process by machine.

Recommendations on the use of work results. Solving the problem of reliable recognition of speech audio streams opens wide prospects for the development of automatic translation systems, automated voice control systems for technical devices, semantic analysis of voice audio streams and many other areas of application.

Keywords: RECOGNITION OF SPEECH AUDIO SIGNALS, NEURAL NETWORK, KOHONEN NEURAL NETWORK, GROSSBERG NEURAL NETWORK.

ЗМІСТ

ВСТУП

Актуальність теми. Проблему автоматичного розпізнавання та розуміння мовлення намагаються вирішити багато десятиліть з часів появи перших мікропроцесорних технічних пристроїв. Але раніше не було достатньо розвинуті методи і технічні засоби для успішного вирішення такої задачі. В той же час, необхідність її розв'язання стимулювалося потребами у військовій та комерційних сферах. Крім того, вирішення задачі автоматичного розпізнавання мовлення є першим кроком для створення більш складніших систем, наприклад, машинного перекладу з однієї мови на іншу, автоматичного аналізу тексту, системи голосового управління технічними пристроями та багато інших.

З розвитком систем штучного інтелекту з'явилося багато нейромереж пов'язаних з розпізнаванням, обробкою та генеруванням голосової інформації, тобто обробки та генерування змістовних мовних аудіо потоків. Прикладом таких систем є Windows, GPT-чати, голосовий пошук в Google та багато інших [1-4]. Проте автоматичне розпізнавання мовленнєвих аудіо потоків та коректна їх інтерпретація потребує подальших досліджень та розробки більш досконалих методів. В даній роботі зроблена спроба здійснити автоматичне розпізнавання мовних аудіопотоків на базі дворівневої нейронної мережі [5-6]. Сучасний стан розвитку методології нейронних мереж та апаратно-технічних засобів дозволяє вирішити та реалізувати дану задачу. При цьому, одним з найбільш важливих етапів побудови таких систем є навчання тренувальними послідовностями. Від успішного вирішення етапу навчання нейронної мережі (НМ) в цілому залежить якість роботи системи розпізнавання. При навчанні НМ на даний момент є дві серйозні проблеми: сильна залежать від стартових параметрів аудіо потоку мовлення і стандартні методи розпізнавання, які не можуть вийти за межі локальних екстремумів функції (метод Баума-Велча або ЕМ-процедура) [6], тобто є методами локальної оптимізації.

Очевидно, що задачі розпізнаванні мовленнєвих сигналів, як зазначалось вище, успішно вирішуються великими корпораціями, які надають можливість користуватися їхніми розробками, проте не розкривають технологій та методів реалізації. Тому для національної науки та компаній важливо володіти власними технологіями, щоб бути конкурентоспроможними та йти в ногу в науковому плані. При цьому слід відмітити, що системи автоматичного розпізнавання мають широкі перспективи використання у найрізноманітніших сферах і є однією з перших задач, які необхідно вирішити для більш складніших систем, наприклад автоматичного перекладу з однієї мови на іншу, здійснення голосового управління, створення систем штучного інтелекту та багатьох інших.

Мета і завдання дослідження. Метою кваліфікаційної роботи є розроблення системи автоматичного розпізнавання мовленнєвої аудіо інформації із застосуванням нейронних мереж. Для досягнення поставленої мети необхідно вирішити наступні завдання.

1. Проаналізувати сучасний стан розвитку систем автоматичного розпізнавання мовленнєвих аудіо потоків (РМАП).

2. На основі проведеного аналізу визначити основні трудності у вирішенні задачі РМАП та на основі сучасного досвіду обрати найкращий шлях для вирішення поставленої задачі.

3. Проаналізувати структури і функціонування нейронних мереж та обґрунтувати обрання структур нейромереж.

4. Розробити структурну та функціональну схему системи РМАП.

5. Розробити структурну схему та алгоритм роботи програми розпізнавання мовленнєвих аудіо потоків.

6. Представити та описати математичний апарат для опрацювання аудіо сигналів.

7. Провести моделювання роботи РМАП.

8. Розробити програмне забезпечення для реалізації методу РМАП.

Об’єкт та предмет дослідження.

Об'єктом дослідження є мовленнєві аудіо потоки.

Предметом дослідження є автоматичне розпізнавання мовленнєвих аудіо потоків на основі нейронних мереж.

Методи дослідження базуються на теорії автоматичного управління, теорії штучного інтелекту, теорії та методах загорткових нейронних мереж, теорії імовірності та математичної статистики, методів програмного та імітаційного моделювання.

Наукова новизна роботи полягає у побудові математичної моделі розпізнавання мовленнєвих аудіо потоків на основі нейронних мереж Кохонена та Гроссберга.

Практична цінність результатів роботи. В результаті досліджень розроблена система автоматичного розпізнавання мовленнєвих аудіо потоків дозволяє правильно перевести мовлення у текст, який набагато простіше опрацьовувати машинним способом.

Напрямки подальшого дослідження. Вирішення задачі достовірного розпізнавання мовленнєвих аудіо потоків відкриває широкі перспективи для розроблення систем автоматичного перекладу, систем автоматизованого управління голосом технічними пристроями, семантичного аналізу голосових аудіо потоків та багато інших сфер застосування.

Публікації:

1. Попик Ю. І., Усенко О. О., Когут Ю. В. Система автоматичного управління роботом маніпулятором. / Збірник матеріалів проблемно-наукової міжгалузевої конференції «Автоматизація та комп'ютерно-інтегровані технології» (АКІТ – 2023), Тернопіль, 2023. С. 72 – 77с.

2. Попик Ю. І., Когут Ю. В., Когут І. Р. Автоматизована система розпізнавання голосової інформації. / Збірник матеріалів науково-практичної конференції молодих вчених, аспірантів та студентів «Кібербезпека та комп'ютерно-інтегровані технології» (КБКІТ – 2023), Тернопіль, 2023. С. 200 – 208с.

1. АНАЛІЗ ОСНОВНИХ ПРОБЛЕМ АВТОМАТИЧНОГО РОЗПІЗНАВАННЯ МОВЛЕННЯ ТА ОГЛЯД ІСНУЮЧИХ СИСТЕМ

1.1 Особливості розпізнавання слів у злитій мові

Головна складність розпізнавання мовленнєвого аудіо сигналу в тому, що він може сильно змінюватися за багатьма параметрами: протяжність, тембр, висота, характерні особливості голосу, що привносяться унікальнісю будови голосового апарату кожної людини та манерою її вимови, різними психологічними і емоційними станами оратора, іншими відмінністями параметрів голосів різних людей з точки зору аналізу аудіо сигналів, зовнішніми акустичними факторами. Навіть мовленнєвий аудіо потік згенерований однією людиною, записаний в той самий момент часу, але на різні пристрої не будуть повністю збігатися. Так само як проголошений однією людиною один і той самий текст в різних промовах буде мати відмінності з точки зору параметрів аудіо сигналу. Тому необхідно знаходити параметри мовленнєвого аудіо сигналу, які б могли з високою достовірністю визначати звуки і відрізняти один звук мови від іншого, в той же час були б слабо чутливими щодо вище описаних варіацій мови. Визначення таких ознак та методів порівняння зі зразками, змушує шукати необхідну форму у параметричному просторі.

Завдання автоматичного розпізнавання мови у загальній постановці задачі полягає у розпізнаванні злитного мовлення довільного змістового напрямку, довільного оратора, в довільному стилі вимови та довільних зовнішніх умовах. Узагальненням цього завдання є «розуміння» промови, у тому числі, враховуючи помилки, що можуть в ній міститися.

Більшість підходів до автоматичного розпізнавання мовлення (АРМ) передбачає, що за допомогою визначення акустичних параметрів та застосування одного із способів пошуку за набором еталонів фонематичних сегментів можна встановити фонематичні ряди. Потім ці ряди можуть бути

використані для проведення лінгвістичного аналізу на вищому рівні виділення окремих слів, фраз і змісту висловлювань. Точне «розуміння» сказаних слів, речень, фраз включає вживання певної лінгвістичної структури у поєднанні з найбільш достовірною звуковою інформацією.

При автоматичному розпізнаванні мови великі труднощі є під час процесів виявлення та ідентифікації деяких груп фонем.

Зараз розпізнавання мовлення складається із трьох моделей. Перша, акустична, відповідає за розпізнавання звуків. Друга, мовна, формує із звуків слова. А третя, пунктуаційна, розставляє розділові знаки. Проте першим кроком попередня обробка звукового сигналу.

У завданнях АРМ в першу чергу необхідно перевести звук у формат, зручний для подачі в мережу. Ця область дуже вимоглива до попередньої обробки вхідних даних. З одного боку, це ускладнює життя дослідникам-початківцям, з іншого — дає поле для творчості.

Сам по собі звук представляється у пам'яті комп'ютера як масив значень, що показують коливання амплітуди за часом. Причому чим більше відліків на секунду ми візьмемо, тим краща буде якість запису (sample rate). Цей параметр зазвичай обчислюється в десятках тисяч точок в секунду або в кГц, і підсумкова доріжка виходить дуже довгою та незручною для роботи. Тому, перш ніж надіслати на вхід алгоритму звук додатково опрацьовують: зокрема, переводять у спектрограму — залежність інтенсивності коливання звуку на конкретній частоті за часом.

Підхід із використанням спектрограм вважається консервативним. Є альтернативи: наприклад, wav2vec. Всупереч тому, що SOTA-рішення (state-of-the-art model) по АРМ зараз використовують wav2vec, для деяких архітектур такий підхід не дає приросту в якості.

Після того, як «сира» доріжка переведена в зручну для нейромережі форму, тобто створена акустична модель, її можна подавати на розпізнавання: отримувати зі звуку розподіл ймовірностей літер за часом.

У більшості підходів спочатку складають фонетичні транскрипції (за

принципом «що чується, те й пишеться»), а потім окремою мовною моделлю покращують результат — виправляють граматичні та орфографічні помилки, забирають із розшифровки зайві літери.

В якості простої акустичної моделі раніше у задачі розпізнавання мови використовувалися марківські моделі [32]. Зараз їх для цього вже ніхто не застосовує – на зміну прийшли нейромережі.

Якщо відкрити приблизно будь-який курс з нейронних мереж, там обов'язково скажуть, що в 2012 році нейромережі перемогли класичні методи завдання класифікації картинок на датасеті ImageNet. В галузі розпізнавання мови така «перемога» відбулася ще 2009 році. Зараз існує вже чимало архітектур, які дають добрі результати.

Наприклад, DeepSpeech2 (SOTA 2018 на датасеті LibriSpeech). Вона була комбінацією двох типів шарів — рекурентних і згорточних. Рекурентні шари дозволяли генерувати продовження фраз з огляду на попередні згенеровані слова, а згорткові відповідали за вилучення ознак із спектрограм. У статті [3] про цю архітектуру вказано, що для навчання використовується CTC-loss: вона дозволяє однаково добре розпізнавати слова «прив-і-і-т» і «привіт», тобто не прив'язуватися до тривалості звуків. До речі, ця функція втрат використовується і в задачах розпізнавання рукописного тексту.

Область Deep Learning [12] дуже швидко розвивається: наприкінці 2018-го ми експериментували зі свіжою DeepSpeech, і в цей же час вийшла стаття про ще одну архітектуру – wav2letter++ від FAIR. Її особливість у тому, що в ній використовуються тільки згорткові шари, тобто немає авторегресивності (з авторегресією ми оглядаємось на минулі згенеровані слова і йдемо послідовно, а це робить нейромережу повільніше).

Використання повністю згорткової архітектури відкрило нові можливості для дослідників. Незабаром з'явилася архітектура Jasper, яка також була fully-convolutional, але також використовувала ідею residual connections, як у ResNet чи трансформерах. Потім вийшла QuartzNet від NVIDIA, заснована на Jasper.

1.2 Аналіз етапів автоматичного розпізнавання мовлення

Проблему РМАП можна вирішувати поетапно (рисунок 1.1). На першому етапі завдання розпізнавання полягає у визначенні загальних характеристик МАП, таких як амплітуда, протяжність, тембр голосу, супутній шум та інші. Цей етап дозволяє відкалібрувати систему автоматичного РМАП. Для другого етапу вирішальне значення має коректне виокремлення фонем: слів, фраз, речень та їх ідентифікація з еталонними значеннями словника

При автоматичному розпізнаванні промови, передусім, слід чітко визначити, чи є сигнал взагалі є фонетичним, тобто мовленнєвим. Як правило, здійснюється розподіл мовного потоку на мікро- і макросегменти. Розмежування між двома макросегментами (фразами синтагмами) носить, зазвичай, дискретний характер, а між двома мікросегментами (субзвуками, звуками, складами) – злитний. Звуки змінюють свої супрасегментні (тривалість, інтенсивність, частота основного тону) та сегментні (спектральні) характеристики відповідно до впливу одиниць інших ярусів. Наприклад, збільшення тривалості голосної в мовному потоці може вказувати на семантичну виділеність слова, положення наголосу щодо цієї голосної, інформацію про попередню і наступну фонеми і т. д. Отже, для передбачення, наприклад, тривалості звуку слід враховувати ряд лінгвістичних факторів.

Функціонування системи відбувається згідно схеми представленої на рисунку 1.1.

Оцифрований мовний сигнал надходить на вхід комп'ютера. Потім сигнал з деяким постійним кроком розбивається на вікна, і для кожного вікна в блоці акустичного аналізу передається вектор значень певних спектральних параметрів, найчастіше кепстральних коефіцієнтів, а також першої та другої дискретних похідних (різниць).

Вектори параметрів послідовно надходять на вхід блоку локального

розпізнавання, що зазвичай, має у своїй основі універсальний монотонний імовірнісний автомат [2,3], що поєднує в собі еталонні імовірнісні автомати всіх слів природної мови, з якими працює система розпізнавання. При надходженні на вхід цього блоку кожного нового вікна аналізу, модифікується орієнтований навантажений граф гіпотез розпізнавання, внаслідок чого до нього додаються нові гіпотези про виголошену послідовність слів мови і видаляються існуючі гіпотези, ймовірність яких є меншою за деякий прийнятий фіксований поріг. Коли надходить останній вектор значень параметрів, у графі залишаються ті гіпотези, які закінчуються на ціле (закінчене) слово мови. Для ефективного функціонування блоку локального розпізнавання істотну роль відіграє вибір фонетичного алфавіту, що потребує окремого дослідження.

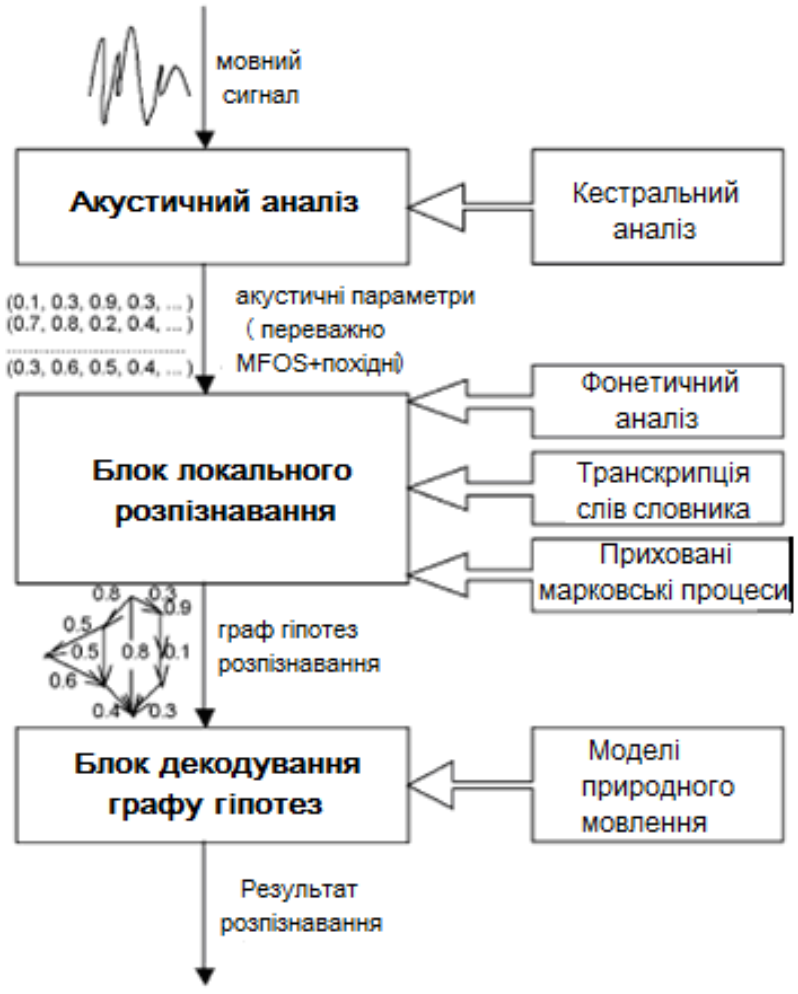


Рисунок 1.1 – Загальна структурна схема системи розпізнавання мовленнєвих аудіо потоків

Фонетичний алфавіт є основою роботи блоку локального розпізнавання мовлення. Як мовилося раніше, кожен трифон мови моделюється монотонним вероятностным автоматом із чотирьох станів. Отже, загальна кількість параметрів автоматів, які необхідно налаштувати в процесі навчання на основі мовного корпусу, лінійно залежить від кількості звуків, тобто від потужності фонетичного алфавіту, і зменшення його розміру призводить до послаблення вимог до обсягу мовної бази даних. З іншого боку, при скороченні алфавіту в ньому можуть бути ототожнені звуки, розрізнення яких може бути істотним у процесі локального розпізнавання. Тому мінімізація розміру фонетичного алфавіту без шкоди якості розпізнавання має бути проведена шляхом ототожнення в алфавіті лише тих звуків, які є найближчими за звучанням з погляду людини.

У роботі [8] показано, що на безлічі автономних ймовірнісних автоматів, за допомогою яких ефективно моделюються звуки та їх поєднання, можна ввести метрику, тісно пов'язану з ймовірністю «плутати» слова при розпізнаванні, тобто з близькістю слів природної мови «на слух» . Ця метрика була ефективно використана авторами під час вирішення завдання оптимального вибору фонетичного алфавіту розробки системи розпізнавання російської мови у межах гранту фірми Intel Corp., США. За допомогою метрики була побудована матриця попарних відстаней між фонемами російської мови, представлених у вигляді автономних автоматів імовірнісних, які були синтезовані на основі російської мовної бази даних. Вдалося показати, що алфавіт із 150 фонемних символів для української мови [5] можна скоротити без потенційної втрати точності при розпізнаванні до 120 символів.

Автоматичний фонетичний транскриптор російських текстів за правилами є важливим елементом розробки системи розпізнавання російської мови. При побудові розміченої частини мовної бази даних як основи навчання параметрів імовірнісних автоматів – еталонів фонем необхідно

побудувати транскрипції текстів, відповідних цим акустичним даним.

В даний момент найскладнішими елементами при побудові систем розпізнавання мовлення є побудова акустичної моделі мови та початкове навчання еталонів слів словника, які найчастіше є ймовірнісними автоматами.

Для налаштування параметрів мовних моделей та еталонів фонетичних одиниць мови як основа для навчання необхідні текстові та мовні бази даних досить великого обсягу. Необхідно ретельно врахувати всі слова і мовні звороти, що зустрічаються в сучасній мові, типи голосів і акцентів, що є у носіїв мови.

Мовні бази даних являють собою безліч записів сказаних різними дикторами слів, фраз, речень. Слова можуть вимовлятися як окремо, і разом; кожна пропозиція у мовному корпусі зазвичай супроводжується фонетичною транскрипцією. Параметри запису можуть бути також різними – від вузькосмугового телефонного запису (моно, частота дискретизації 8 кГц, 8 біт на відлік) та широкосмугового мікрофонного (моно, 22 кГц, 16 біт на відлік) до синхронних багатоканальних записів (телефон + мікрофон, кілька мікрофонів та т.п.) Вузькосмугові бази даних використовуються для створення систем розпізнавання мовлення по телефону, а широкосмугові – для навчання комп'ютерних систем диктування текстів.

Об'єм корпусу характеризується двома важливими параметрами – числом дикторів та загальною тривалістю звучання корпусу. Диктори повинні представляти всі статево-вікові групи, діалекти і т.п. Загальна тривалість корпусу повинна забезпечувати достатню представництво вибірки, що дозволяє зробити якісне навчання параметрів ймовірнісних автоматів.

Важливо, щоб текстова база даних, що становить основу мовного корпусу, містила звані фонетично збалансовані речення, тобто, у яких у середньому рівномірно представлені всі звуки і трифони мови. Крім того, зазвичай тексти включають фрагменти усного діалогу, так і письмову мову. Важливим елементом будь-якої системи, що розпізнає, є розпізнавання послідовностей чисел і цифр, тому частина бази даних, що відповідає набору

чисел, повинна також бути присутньою і бути досить великою за обсягом.

З розвитком мовних технологій і все більшим використанням мобільних пристроїв, виникла ідея застосування мовного управління при побудові мережевих додатків. Для цього необхідно було розробити уніфікований стандарт для інтеграції мовних технологій.

Один із відкритих стандартів на основі XML-мови - VoiceXML (Voice eXtensible Markup Language), перша версія опублікована в травні 2000 р. міжнародним консорціумом World Wide Web (W3 Consortium) - призначений для розробки інтерактивних голосових додатків (Interactive Voice Response, IVR медіаресурси. Мета створення стандарту – привнесення всіх переваг web-програмування у розробку IVR-додатків.

Однак інтерес до багатомодальних додатків, що поєднують розпізнавання мови з іншими формами введення інформації (за допомогою клавіатури, пера або набору цифрових кнопок) спонукав низку компаній, у тому числі Microsoft, підтримати проект SALT Forum (Speech Application Language Tags – теги мови мовних додатків). І тепер навколо SALT та VoiceXML консорціуму W3C формуються два різні табори. Досі компанії не можуть дійти єдиної думки про вибір головного стандарту і зараз обидва напрями розвиваються однаково.

Різні компанії займаються розробкою пакетів до створення мовних додатків, про Software Development Kit (SDK), підтримують той чи інший стандарт. Так компанія Philips створила пакет Speech SDK. Цей пакет підтримує специфікацію Voice XML та виконаний для зв'язку з C/C++ API.

Спільно компаніями CompTek і Philips було створено SpeechPearl — продукт, що представляє собою набір програмних модулів, бібліотек та утиліт для розробки систем розпізнавання мови з підтримкою російської мови для телефонних програм. З іншого боку, корпорація Microsoft розповсюджує свій продукт – Microsoft Speech SDK. Він містить набір компонентів, що описують відповідний програмний інтерфейс Windows Speech API, документацію, вихідні тексти програми-заготівлі (її достатньо доповнити

лише власним алгоритмом розпізнавання), а також системи розпізнавання та перетворення тексту на мову Microsoft Speech Recognition та Microsoft Text-to-Speech [5].

Для використання функцій мовного розпізнавання у різних пристроях, роботах, іграшках розробляються апаратні методи вирішення цієї проблеми. Так, американська компанія Sensory Inc. розробила інтегральну схему Voice Direct™ 364 здійснює дикторозалежне розпізнавання невеликої кількості команд (близько 60) після попереднього навчання [5]. Перед початком експлуатації модуль необхідно навчити всім командам, які у роботі. Команди зберігаються у зовнішню пам'ять як образів розміром 128 байт. Під час роботи образ чергової команди порівнюється з еталонними з пам'яті в нейромережевому модулі і приймається рішення про збіг.

Тайванська технологічна корпорація Primestar Technology Corporation розробила власний чіп VP-2025, призначений для мовного розпізнавання. Цей пристрій здійснює розпізнавання за допомогою нейромережевого методу.

Крім того, американські вчені вирішили створити спеціалізований мікропроцесор для розпізнавання мови. Дослідження в даному напрямку будуть проводитись співробітниками Університету Карнегі-Меллон у Пітсбурзі (Пенсільванія) та Каліфорнійського університету в Берклі. Очікується, що новий мікропроцесор з'явиться протягом двох-трьох років. Причому ефективність розпізнавання мовлення таким чіпом повинна буде в 100-1000 разів перевищити аналогічний показник програмно-апаратних комплексів, що застосовуються сьогодні [5].

1.3 Огляд існуючих методів і систем управління розпізнаванням мовної інформації

Вирішенням питання автоматичного розпізнавання мовленнєвих аудіо потоків інженери і вчені займаються досить довгий період часу. І основні труднощі виникають внаслідок наступних особливостей живого мовлення.

Перша особливість обумовлена високою варіативністю мовного сигналу, викликаною величезною кількістю додаткової (не мовної) інформацією, що є у сигналі. Насамперед, це інформація, що характеризує індивідуальні особливості голосу спікера та стилю його мовлення. Крім того, на звуковий сигнал накладаються поточні акустичні параметри середовища, в якому поширюється мовний сигнал.

Друга особливість полягає в тому, що розпізнавання окремих мовних одиниць: алофонів, фонем, морфем, складів та слів, з високою точністю неможливе на основі інформації, що отримується тільки зі звукового сигналу. Завдання розпізнавання мови вимагає залучення доступної інформації, що дозволяє розділити близькі образи. Крім того, необхідна організація процедури відновлення тієї інформації, якої немає у вихідному мовному сигналі. Це пов'язано, перш за все, з редукованим виголошенням окремих частин слів, а також із втратою інформації через перешкоди в каналах зв'язку. Як показали численні експерименти, деякі ділянки мови можуть бути правильно розпізнані людиною тільки в контексті, який утворює певний образ, наділений семантичним навантаженням [1].

Третя особливість РМАП стосується динамічної природи мовного сигналу. Більшість методів розпізнавання образів орієнтовані на роботу у просторі ознак без врахування часу. Процес розпізнавання мови повинен бути організований так, щоб враховувати порядок проходження в часі одиниць мови, зберігаючи при цьому інваріантність по відношенню до різної тривалості вимови одного й того ж елемента мови.

Основними вимогами, які ставляться до сучасних систем автоматичного розпізнавання мовлення є:

– словники розмірністю у сотні тисяч слів;

– розпізнавання злитного мовлення;

– робота у режимі реальному часі;

– можливість роботи як із попереднім налаштуванням на голос оратора, так і без попередніх налаштуваннь;

– надійність роботи 95-98% для граматично правильних текстів.

Перші пристрої автоматичного розпізнавання мови були аналоговими і використовували порогову логіку, тому вони не мали високої надійності і були вузькоспеціалізованими. Після появи лінгвістичної теорії мови, що представляє мову як похідну фонетичної транскрипції тексту слова, для розпізнавання став використовуватися метод фонетичної сегментації [1, 6], проте згодом з'ясувалося, що це завдання важко піддається точному автоматичному рішенню.

Наступним етапом став розвиток непараметричних підходів, заснованих на мірах близькості безлічі мовних сигналів. Підхід Вінцюка [2], який базується на методі динамічного програмування, дозволив скоротити час обчислення значень функції близькості до еталонних сигналів по відношенню до довжини сигналу з експоненціальної до квадратичнї залежності.

В наслідок того, що основною специфікою методу було нелінійне спотворення часової осі однієї з функцій, що порівнюються, метод отримав назву «динамічної деформації часу» (ДДЧ). Очевидними перевагами методу ДДЧ є його простота і швидкість навчання, а основними недоліками методу – складність обчислення міри близькості, що пропорційний квадрату довжини сигналу і великий обсяг пам'яті, необхідний для зберігання еталонів слів, що пропорційно довжині сигналу і кількості слів у словнику).

Методи, що використовуються в задачах локального розпізнавання мови в даний час, були вперше запропоновані рядом американських вчених: Бейкер-CMU-система «Драгон» і Джелінек-IBM [5] у 1970-ті роки минулого століття. Вони застосували теорію прихованих марківських моделей (ПММ). Приховані марківські моделі являють собою стохастичні процеси – марківські ланцюги [1,5,6] по двох вимірах: по переходах між станами та безліччю стаціонарних процесів у кожному стані ланцюга. Для навчання моделей та обчислення ймовірності спостереження слова на виході ПММ був застосований метод динамічного програмування (алгоритми прямого та зворотного ходу, Баума – Уелча, або ЕМ – алгоритм, Віттербі). Перевагами

методу ПММ є досить швидкий спосіб обчислення значень функції відстані і значно менший, порівняно з методом ДДЧ, обсяг пам'яті, необхідний для зберігання еталонів слів (пропорційно кількості фонем, трифонів тощо.). При цьому, основними недоліками є досить велика складність його реалізації, а також необхідність використання великих фонетично збалансованих мовних баз даних для навчання параметрів ПММ. По суті, методи ДДЧ та ПММ мають дуже багато спільного і можуть вважатися різними реалізаціями одного й того самого підходу.

ПММ, що виникли як узагальнення ланцюгів Маркова, тісно пов'язані з поняттям ймовірносного автомата. Імовірнісні автомати, вперше введені у загальній формі Дж. Карлайлом, являють собою в практичному плані пристрої з кінцевою пам'яттю, що переробляють інформацію з вхідних каналів у вихідні, переходи та виходи яких відбуваються на основі ймовірнісних законів [1]. Приховані марківські моделі є окремим випадком імовірнісних автоматів, а саме, імовірнісними автоматами без входу. ПММ у системах розпізнавання мови, мають додаткову властивість, яка полягає у тому, що при кожному такті роботи автомата перехід здійснюється у стан із тим самим чи більшим номером.

Такі моделі, запропоновані вперше Бакісом, називаються лівоправими (left-right), або моделями Бакіса. У [1] запропоновано називати відповідні таким моделям імовірнісні автомати – монотонними.

Відповідно до [1], метод прихованих марківських моделей можна викласти мовою імовірнісних автоматів. Окремий випадок ймовірнісних автоматів – ініціальні (задані початковий та фінальний стани) автономні (без входу) монотонні автомати Мура, в якому вихід і перехід у наступні стани здійснюються незалежно. З метою прискорення обчислень часто замість обчислення ймовірності по всіх ланцюжках станів, що ведуть від початкового стану до фінального, знаходять ланцюжок з максимальною ймовірністю для даного вихідного слова і цю ймовірність вважають шуканою ймовірністю.

Незважаючи на те, що теоретичне обґрунтування такого підходу не

дуже чітке, він дає значний виграш у часі, що зумовлює популярність цього методу.

Окремим і, мабуть, найнетривіальнішим завданням на етапі локального розпізнавання мови є завдання синтезу (навчання параметрів) монотонного ймовірнісного автомата.

Еталонні імовірні автомати для слів природної мови складаються шляхом послідовного з'єднання відповідних еталонних автоматів трифонів, при цьому фінальні стани всіх таких автоматів, крім останнього, склеюються з першим станом наступного трифона.

Навчання імовірнісного автомата полягає в тому, щоб при заданій кількості станів автомата і, можливо, деякому початковому наближенні значень параметрів автомата відповідало заданому кінцевому набору вихідних слів навчальної вибірки.

Питання здійсненності припущень, що лежать в основі застосування методу прихованих марківських моделей, є відкритими. Тим не менш, практика показує, що, незважаючи на неадекватність моделі, цей метод дає добрі результати.

В основі будь-якої мовної технології лежить так званий engine або ядро програми – набір даних і правил, за якими здійснюється обробка даних. Залежно від призначення цього ядра розрізняють TTS та ASR engine. TTS (Text-to-Speech) engine надає можливість синтезу промови за текстом, а ASR (Automatic Speech Recognition) engine – для розпізнавання промови.

Існує кілька великих виробників, що займаються створенням ядер ASR і серед них такі компанії, як SPIRIT, Advanced Recognition Technologies, IBM.

Компанія SPIRIT займається створенням програмних засобів для цифрової телефонії, стиснення мови, ідентифікації мовця, технологій VoIP і GPS [6]. ASR engine від SPIRIT розроблений для розпізнавання мовних команд і застосовується в різних програмах, таких як голосове керування пристроями, голосовий набір в hands-free пристроях, введення персональних ідентифікаційних кодів (PIN) у системах безпеки. Дане ядро вбудовується в

будь-які платформи DSP або RISC і поставляється у вигляді об'єктного коду.

Корпорація IBM вже понад 30 років займається питаннями автоматичного розпізнавання мови та досягла у цій галузі великих успіхів. Так, компанія ProVox Technologies на основі програмного ядра ViaVoice® від IBM [5] створила систему для диктування звітів лікарів-радіологів VoxReports. За результатами тестувань, дана система з точністю 95-98% розпізнає злиту мову нормального темпу (до 180 слів за хвилину) незалежно від диктора. Однак, словник системи обмежений набором специфічних медичних термінів.

Opera Software домовилася з IBM про інтеграцію до браузера Opera технології розпізнавання мови Embedded ViaVoice [5]. Використання Embedded ViaVoice дозволить користувачам керувати браузером не лише за допомогою миші та клавіатури, але й голосом.

Технологія розпізнавання мовлення все більше розвивається і застосовується у засобах мобільного зв'язку. Так компанія Advanced Recognition Technologies створила систему smARTspeak NG, що вбудовується у мобільні телефони. Дана система smARTspeak NG застосовується у смартфонах від Siemens та Panasonic стандарту TDMA у США та інших.

Програмна розробка білоруської компанії «Сакрамент» – Sakrament ASR Engine, що використовує технології розпізнавання мовлення, розрахована на застосування у різних апаратних системах та програмних додатках. Заявлені характеристики: точність розпізнавання 95-98%; дикторонезалежність; мовонезалежність; розпізнавання злитної мови у вигляді фраз та невеликих речень. Однак у цій системі немає можливості навчання – додаткові словники створюються на замовлення самою компанією «Сакрамент».

В даний час ринок програмних розпізнавачів мови представлений безліччю програм. Розглянемо найвідоміші з них.

Dragon NaturallySpeaking Preferred фірми Dragon Systems [5] є першою, що наблизилася до того, щоб відповідати заявленим характеристикам.

Загалом вона дуже підходить для досягнення заявленої безпомилковості розпізнавання – 95%. Хоча пакет Dragon і поступається деяким конкурентам у томуінтерфейсі та форматуванні, проте він перевершує всіх в основному – можливості з першого разу правильно розпізнавати сказані слова. Спочатку цей пакет не працював з українською мовою, але з годом перелік мов розширився.

Розпізнавання мовлення може застосовуватися як для введення тексту чи подачі команд, але й більш специфічних цілей. Так компанія «Центр Мовних Технологій» розробляє програмні продукти, технології та зразки техніки для підрозділів МВС, МЮ, МНС, МО, служб екстреної допомоги, центрів обробки викликів та інших користувачів, у діяльності яких особливе значення надається реєстрації та обробці мовної інформації.

Компанією створені такі додатки: «ІКАР Лаб» – інструментальний комплекс криміналістичного дослідження фонограм мовлення, «Трал» – автоматизований комплекс розпізнавання дикторів у фонограмах телефонних переговорів, «Територія» – автоматизована система діагностики діалектів та акцентів російської мовлення.

Німецький інститут DFKI, що займається розробками в галузі штучного інтелекту, розробив систему, названу Verbmobil, здатну перекладати розмовну промову з німецької на англійську або японську і назад безпосередньо сказану в мікрофон. Система виконана у вигляді незалежного сервера Verbmobil Server. Завдяки цьому Verbmobil вдалося зв'язати з мережею мобільних телефонів стандарту GSM. Тепер різномовні абоненти, підключившись до Verbmobil Server, можуть спілкуватися один з одним безпосередньо, приймаючи вже перекладену мову, при цьому Verbmobil автоматично налаштовується на мову мовника. За даними експериментів, точність перекладів становить 90%, що було перевірено на 25 000 тестових фразах.

Для використання функцій мовного розпізнавання у різних пристроях, роботах, іграшках розробляються апаратні методи вирішення цієї проблеми.

Так, американська компанія Sensory Inc. розробила інтегральну схему Voice Direct™ 364 здійснює дикторозалежне розпізнавання невеликої кількості команд (близько 60) після попереднього навчання [5]. Перед початком експлуатації модуль необхідно навчити всім командам, які у роботі. Команди зберігаються у зовнішню пам'ять як образів розміром 128 байт. Під час роботи образ чергової команди порівнюється з еталонними з пам'яті в нейромережевому модулі і приймається рішення про збіг.

Тайванська технологічна корпорація Primestar Technology Corporation розробила власний чіп VP-2025, призначений для мовного розпізнавання. Цей пристрій здійснює розпізнавання за допомогою нейромережевого методу.

Крім того, американські вчені вирішили створити спеціалізований мікропроцесор для розпізнавання мови. Дослідження в даному напрямку будуть проводитись співробітниками Університету Карнегі-Меллон у Пітсбурзі (Пенсільванія) та Каліфорнійського університету в Берклі. Очікується, що новий мікропроцесор з'явиться протягом двох-трьох років. Причому ефективність розпізнавання мовлення таким чіпом повинна буде в 100-1000 разів перевищити аналогічний показник програмно-апаратних комплексів, що застосовуються сьогодні [5].

Найбільш сучсні системи розпізнавання мовлення базуються на використанні нейронних мереж. Найбільш відомі на сьогоднішній день є ряд зарубіжних сервісів, коротко описані нижче.

«Google Cloud Speech-to-Text – це сучасний інструмент для автоматичного перетворення мовлення в текст і транскрипції. Це корисні сервіси мовлення від Google, що дозволяє розробникам використовувати автовідповідачі в кол-центрах, дозволяє IoT-пристроям спілкуватися з користувачами та перетворювати текстові повідомлення в голосовий формат.

Speech-to-Text, який раніше називався Cloud Speech API, був вперше випущений у 2016 році. За даними Google, у перші роки його роботи використання API подвоювалося кожні шість місяців. Це рішення базується

на передових алгоритмах нейронної мережі глибокого навчання Google для автоматичного розпізнавання мови (ASR).» [2]

Є можливість швидко розгорнути ASR у хмарі за допомогою API або навіть локально за допомогою локального перетворення мови на текст, що інтегрує технології розпізнавання мовлення Google у локальне рішення користувача. Для забезпечення необхідних правил розміщення даних та відповідності вимогам, користувач можете взяти під контроль свою інформаційну інфраструктуру, одночасно отримуючи доступ до технології розпізнавання мовлення з високозахищеними мовними даними.

Технологія розпізнавання мови існує вже кілька десятиліть, але зараз ми починаємо бачити справжній потенціал цієї технології. З появою GPT4, потужної системи обробки природної мови, точність розпізнавання мови різко збільшилася.

GPT4 – це четверта ітерація системи генеративного попередньо навченого трансформатора (GPT), розроблена OpenAI. GPT4 – це просунута мовна модель, яка навчається на великому масиві текстів. Це означає, що він може розпізнавати шаблони у тексті та генерувати новий текст на основі цих шаблонів. На додаток до цього GPT4 був розроблений для розпізнавання та інтерпретації природної мови, що означає, що він може розуміти значення слів, що вимовляються.

Це робить GPT4 придатним для використання у розпізнаванні мови. Використовуючи можливості GPT4, системи розпізнавання мови можуть точно інтерпретувати слова і фрази, що вимовляються, навіть у шумній обстановці. Крім того, GPT4 також можна використовувати для створення природнішого звучання мови, забезпечуючи більш реалістичний досвід при використанні цих систем.

GPT4 також здатний навчатися на своїх помилках та адаптуватися до різних сценаріїв. Це означає, що точність систем розпізнавання мовлення може бути покращена з часом, оскільки GPT4 навчається на своїх помилках і стає кращим у розпізнаванні розмовної мови.

Потенціал GPT4 для розпізнавання мовлення є очевидним. Використовуючи цю передову технологію, можна значно підвищити точність розпізнавання мови, надаючи користувачам природніший і точніший досвід при використанні цих систем. Оскільки GPT4 продовжує розвиватися та вдосконалюватися, ми можемо очікувати ще більшого підвищення точності у найближчому майбутньому.

GPT4, розроблений OpenAI, є останнім поколінням алгоритмів обробки природної мови (NLP). Його подають як революційну технологію, яка може зробити революцію в багатьох різних додатках, включаючи додатки для чат-ботів.

По-перше, GPT4 дуже ефективний в розумінні природної мови. Він використовує різні методи, щоб зрозуміти контекст розмови та генерувати відповідні відповіді. Це робить його ідеальним для використання в додатках чат-ботів, де розмова має бути природною.

По-друге, GPT4 може генерувати відповіді в режимі реального часу. Це дозволяє чат-боту швидко реагувати на вхідні повідомлення, роблячи спілкування більш плавним та природним.

По-третє, GPT4 добре масштабується. Це дозволяє використовувати GPT4 у програмах, які потребують великої кількості діалогів або відповідей.

Зрештою, GPT4 відносно дешевий. Це робить його привабливим варіантом для підприємств, яким потрібне недороге рішення для програми чат-бота.

В цілому, GPT4 – вражаюча технологія, яка може зробити революцію в багатьох додатках. Оскільки технологія продовжує вдосконалюватися, цілком можливо, що GPT4 стане ще більш привабливим варіантом для бізнесу.

Нейронні мережі Microsoft тепер розпізнають людський голос так само добре, як і люди. У доповіді команди дослідників у галузі наукового інтелекту Speech & Dialog сказано, що система розпізнавання мови тепер помиляється так само часто, як професійні стенографісти. У деяких випадках система здатна робити менше помилок.

Під час тестів коефіцієнт помилкових слів (WER) склав 5,9%, що нижче за попередній результат 6,3%, про який Microsoft повідомила минулого місяця. Це найнижчий результат коли-небудь зареєстрованих розпізнавань живого мовлення. Команда не вважає це проривом в алгоритмі або даних, але в ретельному налаштуванні існуючих архітектур штучного інтелекту. Основна складність полягає в тому, що навіть якщо звукова доріжка хорошої якості і не містить сторонніх шумів, алгоритм має боротися з різними голосами, перервами, коливаннями та іншими нюансами живої мови людини.

Щоб перевірити, наскільки алгоритм здатний повторити людські здібності, Microsoft чистоти експерименту найняла стенографістів з боку. У компанії вже була готова правильна стенограма аудіофайлу, запропонована фахівцям. Стенографісти працювали у два етапи: спочатку одна людина передруковувала аудіо-фрагмент, а потім друга слухала та виправляла помилки у розшифровці стенограми. На основі правильної стенограми для стандартизованих тестів фахівці, розшифровуючи запис розмови на конкретну тему, спрацювали на 5,9%, а результат розшифровки вільного діалогу показав 11,3% помилок. Після 2 000 годин навчання людської мови, за цими ж аудіофайлами система Microsoft набрала 5,9% та 11,1% помилок відповідно. Це означає, що комп'ютер тепер може розпізнавати слова у розмові так, якби він був людиною. При цьому команда виконала мету, яку поставила перед собою менше року тому, а результат значно перевершив очікування.

Тепер Microsoft збирається повторити такий самий результат у шумній обстановці. Наприклад, під час руху шосе або на вечірці. Крім того, компанія планує зосередити свою увагу на більш ефективних способах допомогти технології розпізнавати окремих спікерів, якщо вони говорять одночасно, та переконатися, що AI добре працює з великою кількістю голосів незалежно від віку та акценту. Реалізація цих можливостей у майбутньому має вирішальне значення і виходить за рамки простої стенографії.

Щоб досягти таких результатів, дослідники використали власну розробку компанії – обчислювальну мережу Toolkit. Можливість цього нейромережевого інструментарію швидко обробляти навчальні алгоритми на кількох комп'ютерах, які працюють під управлінням графічного процесора, значно покращила швидкість, з якою вони могли проводити дослідження, і, зрештою, досягти людського рівня.

Такий рівень точності виявився можливим завдяки використання поєднання трьох варіантів згорткової нейронної мережі (таблиця 1.1). Першою з них стала архітектура VGG, що відрізняється великою кількістю прихованих шарів.

Таблиця 1.1 – порівняльні характреристики архітектур згорткових нейронних мереж в WINDOWS системі розпізнавання мовлення

VGG Net (85M Parameters)	**Residual-Net (38M Parameters)**	**LACE (65M Parameters)**
14 weight layers	49 weight layers	22 weight layers
40x41 input	40x41 input	40x61 input
3 – conv 3x3, 96	3 – [conv 1x1, 64 conv 3x3, 64 conv 1x1, 256]	5 – conv 3x3, 128
Max pool	4 – [conv 1x1, 128 conv 3x3, 128 conv 1x1, 512]	5 – conv 3x3, 256
4 – conv 3x3, 192	6 – [conv 1x1, 256 conv 3x3, 256 conv 1x1, 1024]	5 – conv 3x3, 512
Max pool	3 – [conv 1x1, 512 conv 3x3, 512 conv 1x1, 2048]	5 – conv 3x3, 1024
4 – conv 3x3, 384	Average pool	1 – conv 3x4, 1
Max pool	Softmax (9000)	Softmax (9000)
2 – FC – 4096		
Softmax (9000)		

У порівнянні з мережами, які використовувалися раніше для розпізнавання зображення, ця мережа застосовує невеликі, глибші фільтри (3x3), а також використовує до п'яти рівнів згортки перед об'єднанням. Друга мережа змодельована на архітектурі ResNet, яка додає магістральні з'єднання. Єдина відмінність у тому, що розробники застосували пакетну нормалізацію перед тим, як обчислити ReLU. Остання мережа згортання в списку — LACE (рисунок 1.2).

Це варіант нейронної мережі з тимчасовою затримкою, у якому кожен вищий рівень – нелінійне перетворення зважених сум вікон кадрів нижнього рівня.

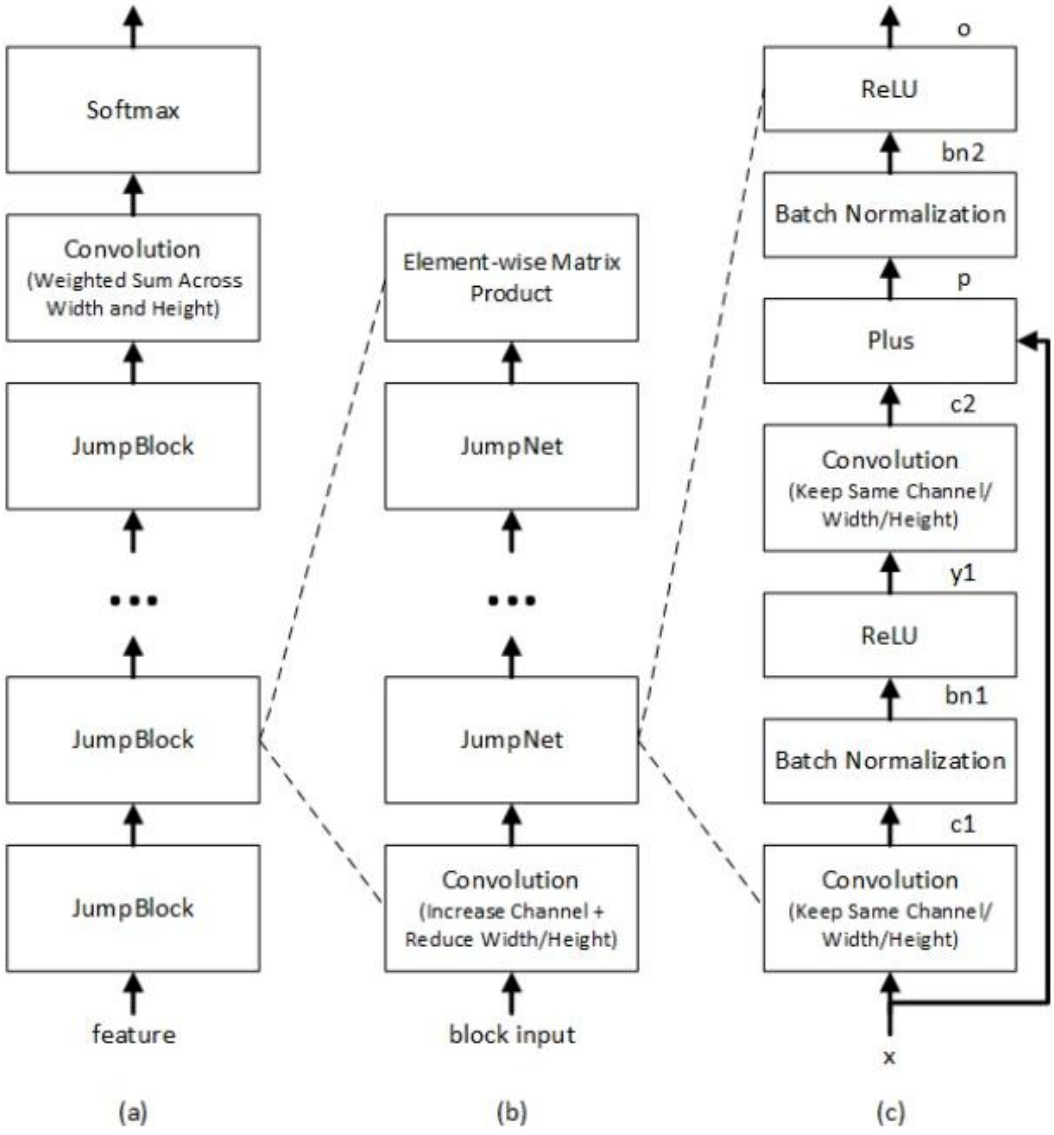


Рисунок 1.2 – Архітектура LACE Network

Іншими словами, кожен вищий рівень використовує ширший контекст, ніж нижні рівні. Нижні рівні фокусуються на вилученні простих локальних структур, тоді як вищі рівні витягують складніші структури, які покривають ширші контексти.

2. РЕАЛІЗАЦІЯ МЕТОДУ ТА СТРУКТУРИ СИСТЕМИ РОЗПІЗНАВАННЯ МОВЛЕННЄВИХ АУДІО ПОТОКІВ НА ОСНОВІ НЕЙРОМЕРЕЖ

2.1 Характеристики сучасних нейропакетів

В даний час відома велика кількість нейропакетів, що випускаються рядом фірм та окремими дослідниками і дозволяють конструювати, навчати та використовувати нейронні мережі для вирішення практичних завдань. Розглянемо декілька нейропакетів, призначених для реалізації на персональних комп'ютерах у різних операційних середовищах, за ступенем їхньої універсальності, а також з погляду простоти використання та наочності подання інформації.

Нейропакет NeuroSolutions призначений для моделювання великого набору нейронних мереж. Основна його гідність полягає в гнучкості: крім традиційних нейромережевих парадигм (повнозв'язкових і багатошарових СР, самоорганізуються карт Кохонена) нейропакет включає потужний редактор візуального проектування нейронних мереж, що дозволяє створювати будь-які нейронні структури і алгоритми їх навчання, а також вводити власні критерії. NeuroSolutions має гарні засоби візуалізації структур, процесів та результатів навчання та функціонування нейронних мереж. Це ставить даний нейропакет на рівень CAD-систем (систем автоматизованого проектування) проектування та моделювання СР.

Пакет призначено для роботи Windows. Крім засобів взаємодії з операційною системою (OLE), нейропакет має генератор вихідного коду і дозволяє використовувати зовнішні модулі при створенні та навчанні нейронної мережі. Пакет підтримує програми, написані мовою C++ для компіляторів Microsoft Visual C++ і Borland C++, і навіть як DLL-коду. Таким чином, NeuroSolutions є гнучкою відкритою системою, яку можна за необхідності доповнювати та модифікувати. Пакет містить вбудовану

макромову, що дозволяє робити практично будь-яке налаштування під конкретне завдання.

У пакеті реалізується великий перелік нейронів, включаючи зважений суматор (нейрон першого порядку), нейрони вищих порядків (з перемноженням входів), а також безперервний нейрон, що інтегрує. Функція активації нейрона може бути обрана з п'яти стандартних (кусково-лінійна, функція знака та три типи сигмоїдальних) функцій, а також задана користувачем. Зв'язки між нейронами задаються довільно на етапі проектування та можуть бути змінені у процесі роботи. Підтримуються всі типи зв'язків: прямі, перехресні та зворотні. При цьому добре реалізована схема організації зв'язків: можна задати один векторний зв'язок із заданою ваговою матрицею, а не набір скалярних зв'язків із ваговими коефіцієнтами.

Нейропакет NeuroSolutions містить потужні засоби для організації навчальних вибірок. Вбудовані конвертори даних підтримують графічні зображення у форматі BMP, текстові файли з числовими або символьними даними, а також функції безперервного аргументу (наприклад, часу), задані в аналітичному вигляді або як вибірка значень. Нейропакет дозволяє використовувати будь-які зовнішні конвертори даних.

На етапі навчання можна використовувати широке коло критеріїв навчання, як дискретних, і безперервних. Крім цього, можна вводити власні критерії. Можна використовувати як вбудований алгоритм навчання типу back-propagation чи дельта-правила, і використовувати власний. Система візуалізації процесу навчання дозволяє проводити аналіз зміни ваг безпосередньо у процесі навчання та вносити корективи. Може бути введена шумова характеристика як під час тестування, і під час навчання нейронної мережі. Можна встановити адитивний білий шум, шум довільної природи, а також будь-який заданий тип шуму (наприклад, білий мультиплікативний). Neurosolutions містить генератор (майстер) стандартних нейромережевих архітектор Neural Wizard, за допомогою якого швидко задається архітектура, підбираєте навчальну вибірку, критерії та методи навчання нейронної мережі.

Нейропакет NeuralWorks Professional II/Plus є потужним засобом для моделювання нейронних мереж. У ньому реалізовано 28 нейронних парадигм, а також велику кількість алгоритмів навчання. Додатковий модуль UDND (User Define Neural Dynamics) дозволяє створювати власні нейронні структури.

Як і NeuroSolutions, NeuralWorks Professional має хорошу систему візуалізації даних: структури нейронної мережі, зміни помилки навчання, зміни ваг та їх кореляції у процесі навчання. Останнє є унікальною властивістю пакета та корисна під час аналізу поведінки мережі.

У NeuralWorks Professional можна інтегрувати зовнішні модулі. Він має вбудований генератор коду, який підтримує компілятор Microsoft Visual C++.

Спосіб подання інформації трохи відрізняється від NeuroSolutions.

Нейропакет Process Advisor призначений на вирішення завдань управління динамічними процесами (зокрема, технологічними процесами). Проте може вважатися універсальним нейропакетом. У ньому реалізована лише багатошарова нейронна мережа прямого поширення, що навчається за допомогою модифікованого алгоритму зворотного розповсюдження помилки. У пакет введено можливість роботи з вхідними сигналами як із функціями часу, а чи не дискретним набором точок. Таку можливість крім Process Advisor має лише NeuroSolutions. Крім того, нейропакет Process Advisor дозволяє керувати зовнішніми апаратними контролерами, що підключаються до комп'ютера. Саме ці дві особливості роблять нейропакет Process Advisor чудовим.

Нейропакет NeuroShell 2 є однією з трьох програм, що входять до складу пакету The Al Trilogy і є універсальним нейропакетом для моделювання декількох найбільш відомих нейронних парадигм: багатошарових мереж, мереж Кохонена і т.д.

NeuroShell 2 сильно програє в порівнянні з NeuroSolutions та NeuralWorks. Він має багато дрібних недоліків, що суттєво уповільнюють підготовку та роботу в середовищі нейропакета. Крім недостатньо

продуманого інтерфейсу, нейропакет NeuroShell має і ускладнену систему візуалізації даних. Через відсутність єдиного інтегрального контролю даних у процесі навчання або роботи нейронної мережі часто доводиться перемикатися з одного режиму до іншого, що незручно у використанні.

Для NeuroShell характерна жорстка послідовність дій під час роботи з нейронною мережею. Це зручно для користувачів-початківців. Однак, для того, щоб внести невелику зміну, доводиться виконувати наново всю послідовність дій.

NeuroShell надає хороші засоби обміну даними з іншими програмами. Він забезпечує обмін даними, представленими у текстовому бінарному вигляді, а також у найбільш популярних фінансових форматах MataStock та DowJones. Нейро-пакет має генератор вихідного коду мовами Visual C.

Нейропакет BrainMaker Pro є простим нейропакетом для моделювання багатошарових нейронних мереж, які навчаються за допомогою алгоритму зворотного розповсюдження помилки. Основною його перевагою є велика кількість параметрів налаштування алгоритму навчання. В іншому BrainMaker Pro поступається NeuroSolutions і NeuralWorks, особливо, в наочності представленої інформації та простоти інтерфейсу.

Нейросимулятор SNNS 4.1 дозволяє емулювати роботу досить великої кількості алгоритмів нейронних мереж і є найбільш універсальним і багатофункціональним серед програмних нейросимуляторів, що вільно розповсюджуються.

Програмний пакет MATLAB надає найповніші можливості щодо дослідження властивостей алгоритмів навчання IHM. Neural Network Toolbox, що входить до його складу, є пакетом програм, орієнтованим на вирішення широкого спектру завдань з використанням нейромережевих алгоритмів. У ньому передбачена реалізація 15 різновидів нейронних мереж, а також можливість створення користувацьких мереж практично будь-якої конфігурації.

Нейросимулятор Trajan – у пакеті реалізовані алгоритм зворотного розповсюдження, алгоритм сполучених градієнтів, Левенберга-Марквардта, швидкого поширення, Delta-Bar-Delta, що дозволяє використовувати версію з освітньою метою.

2.2 Аналіз ефективності використання нейромереж для створення системи розпізнавання мовлення

Результати дослідження штучних нейронних мереж [8,9] свідчать про широкі можливості їх використання у різноманітних сисемах обробки інформації. У зв'язку з універсальністю, гнучкістю та ефективністю цього способу обробки інформації в порівнянні з існуючими на сьогоднішній день іншими традиційними методами, він може використовуватися для вирішення широкого кола задач найрізноманітнших сфер людської діяльності, для яких можливо формалізуати певним чином вхідні і вихідні параметри.

Розпізнавання образів, одним із додатків якої є розпізнавання мови, є одним із завдань, що успішно розв'язуються нейромережами.

Здійснювати класифікаціє є однєю з основних функцій нейромережі за допомогою якої вона вирішує задачі. При цьому нейромережа здатна сама вчитися класифікувати об’єкти бе зовнішнього втручання. Проте внутрішні закони, правила чи залежності, яким чином це відбувається залишаються невідомими навіть для розробника цієї нейромережі.

Що стосується мовленнєвих аудіо сигналів, то їх можна представити у вигляді вектора в деякому параметричному просторі, а потім цей вектор може бути запам'ятований нейромережею. Однин з типів нейромережі, що навчається без вчителя є карта ознак Кохонена, що самоорганізується. Вона характеризується тим що набір вхідних сигналів складає нейронні ансамблі, що є образами цих сигнали. Такий підхід приводить до статистичного усереднення сигналів, що дозволяє уникнути проблем, пов’язаних з варіативністю мовлення різних спікерів. Так само як і будь-які інші

нейромережеві алгоритми, дана нейромережаздатна здійснювати паралельну обробку інформації, тобто одночасно задіяні всі нейрони. Завдяки цьому підвищується швидкість обробки інформації, як наслідок – швидкість розпізнавання. Таким чином, декілька ітерацій роботи нейромережі достатньо для розпізнавання.

Далі, на основі нейромереж легко будуються ієрархічні багаторівневі структури, при цьому зберігається їхня прозорість (можливість їх роздільного аналізу). Оскільки власне мова є складовою, тобто. розбивається на фрази, слова, букви, звуки, те й систему розпізнавання мови логічно будувати ієрархічну.

Нарешті, ще однією важливою властивістю нейромереж є гнучкість архітектури. Автоматичне створення алгоритмів – це мрія вже кілька десятиліть. Але створення алгоритмів мовами програмування поки що під силу лише людині. Звичайно, створені спеціальні мови, що дозволяють виконувати автоматичну генерацію алгоритмів, але й вони не набагато спрощують це завдання. У нейромережах генерація нового алгоритму досягається простою зміною її архітектури. При цьому можна отримати абсолютно нове рішення задачі. Ввівши коректне правило відбору, що визначає, краще чи гірше нова нейромережа вирішує завдання, і правила модифікації нейромережі, можна отримати нейромережу, яка вирішить завдання правильно.

Нейросети дозволяють створити досить універсальну автономну систему з можливістю адаптації, наділяючи її здатністю вчитися.

При цьому особливостями та перевагами нейромереж є наступні характеристики.

а) Розробка нейромережі полягає лише у побудові її архітектури.

Під час створення нейромережі розробник створює тільки функціональну структуру, без наповнення її інформацією взагалі або мінімальним об’ємом. Необхідна інформація накопичується системою у процесі навчання.

б) Система сама контролює свої дії з за рахунок корекції на наступній ітерації.

Така властивість передбачає наявність зворотного зв'язку у системі за принципом: ДІЯ – РЕЗУЛЬТАТ – КОРЕКЦІЯ, і повторення при необхідності наступної ітерації. Така система корекції дій для досягенння результату поширенна у складних біологічних організмах і використовуються на всіх рівнях.

в) Забезпечується можливість накопичення знань про об'єкти сфери інтересів.

«Знання про об'єкт – це здатність маніпулювати його образом в пам'яті, тобто, кількість знань про об'єкт визначається не тільки набором його властивостей, але ще й інформацією про його взаємодію з іншими об'єктами, поведінці при різних впливах, знаходженні в різних станах його поведінці у зовнішньому оточенні і т.д.»[15-18] Це властивість наділяє систему можливістю абстрагування від реальних об'єктів, тобто, можливістю аналізувати об'єкт за його відсутності, цим відкриваючи нові можливості у навчанні.

г) Забезпечується автономність роботи системи.

«При інтеграції комплексу процесів, які система здатна здійснювати, з комплексом датчиків, що дозволяють контролювати свої дії та зовнішнє середовище, наділена вищенаведеними властивостями система буде здатна взаємодіяти із зовнішнім світом на досить складному рівні, тобто, адекватно реагувати на зміну зовнішнього оточення (звісно, якщо це буде закладено в систему на етапі навчання). Здатність коригувати свою поведінка залежно від зовнішніх умов дозволить частково чи повністю усунути необхідність контролю ззовні, тобто, система стане автономною.» [15-18]

д) Реалізується самонавчання системи

Для вивчення особливостей та самої можливості самонавчання, системи розпізнавання та синтезу мовлення були об'єднані в єдину систему, що дозволило реалізувати цей процес самонавчання. Таке поєднання є однією

із ключових властивостей створюваної моделі, оскільки дозволяє системі здійснювати і синтез мовленєвих аудіо сигналів і розпізнавати їх під час навчання.

е) Створюється можливість переведення образів, що запам'ятовуються, в новий параметричний простір з набагато меншою розмірністю за рахунок того, що кількість параметрів моделі синтезу мови набагато менша кількості первинних ознак моделі, які є несуттєвими для розпізнавання мови.

Одним з ефективних застосувань нейромережевого підходу до обробки мовленнєвих аудіо сигналів є двоканальна нейромережева система розпізнавання мовленнєвих команд, описана в [10]. У цій роботі пропонується реалізація першого каналу як нейромережевого пофонемного розпізнавача, а другого – як нечіткого класифікатора цілісних патернів (слів).

2.3 Модель нейромережі для розпізнавання мовлення

Структуру нейромережа для розпізнавання мовлення є досить простою і складається з трьох рівнів (рисунок 2.1): вхідний шар, символьний шар і ефекторний шар. Кожен нейрон наступного шару має зв’язки з усіма нейронами попереднього шару. Функція передачі даних у всіх шарах має лінійну залежність, у вхідному шарі моделюється ефект конкуренції.

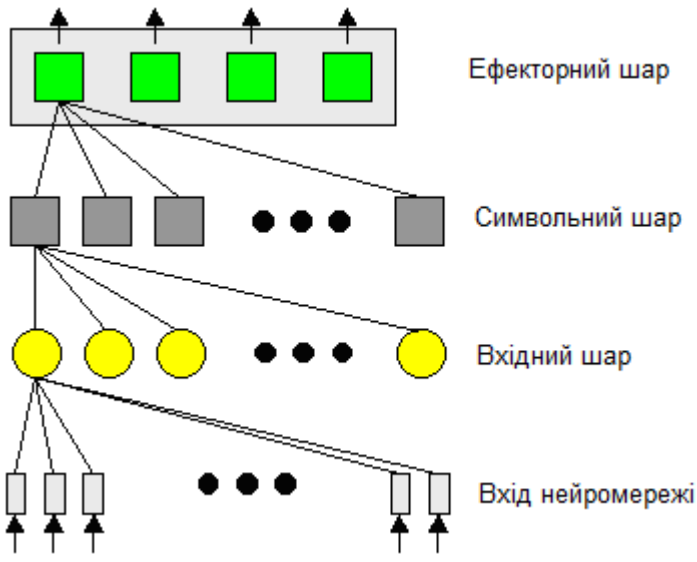


Рисунок 2.1 – Загальна архітектура нейромережі для розпізнавання мовлення

Вхід нейромережі це вхідні мовленнєві аудіо сигнали, після початкової обробки, описаної вище.

Вхідний шар безпосередньо не здійснює маніпуляції над вхідними даними, а лише розподіляють їх для подальшої обробки в нейромережі. Для організації цього шару нейромереж обрано карти Кохонена, що самоорганізується і навчається без вчителя.

Символьний шар включає нейрони, кожен з яких відповідає певному символаму алфавіту. Цей шар здійснює генерацію символів при розпізнаванні аудіо сигналів. Він організований як нейромережа Гроссберга, що навчається з учителем.

Ефекторний шар отримує сигнали від попереднього символьного шару і також є шаром Гроссберга. Вихідними даними цього шару є вектор ефекторів. Цей шар дозволяє зіставити кожному нейрону символьного шару (а отже, і кожному символу алфавіту) деякий вектор ефекторів (а, отже, і певний синтезований звук). Навчання цього шару відбувається так само як і символьного.

Шар Кохонена – це мережа, яка складається з M нейронів, що утворюють прямокутні решітки на площині, як показано на малюнку 2.2.

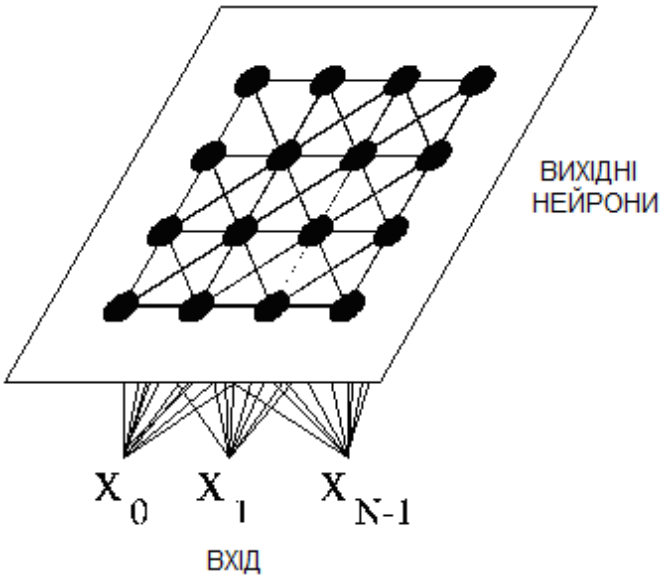


Рисунок 2.9 – Топологія нейронної мережі Кохонена

Елементи вхідних сигналів подаються на входи всіх нейронів мережі. У процесі роботи алгоритму налаштовуються синоптичні ваги нейронів.

Вхідні сигнали – вектори дійсних чисел – послідовно висуваються мережі. Бажані вихідні сигнали не визначаються. Після того, як було пред'явлено достатньо вхідних векторів, синаптичні ваги мережі визначають кластери. [8, 9] Крім того, ваги організуються так, що топологічно близькі вузли чутливі до схожих зовнішніх дій (вхідних сигналів).

Для реалізації алгоритму необхідно визначити міру сусідства нейронів (міру близькості). На рисунку 2.3 показані зони топологічного сусідства нейронів на карті ознак у різні моменти часу. $NE_j(t)$ – безліч нейронів, які вважаються сусідами нейрона j на момент часу t. Зони сусідства зменшуються з часом.

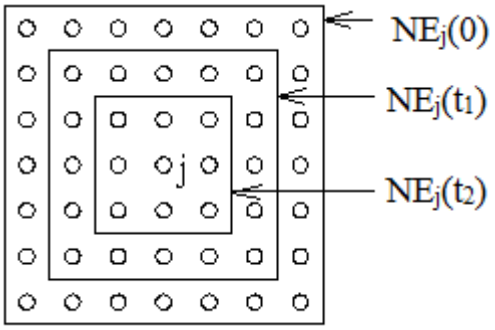


Рисунок 2.3 – Зони топологічного сусідства на карті ознак у різні моменти часу

У штучній нейронній мережі Кохонена нейрони вихідного шару називають кластерними елементами, їх кількість визначає максимальну кількість груп, на які система може розділити вхідні дані. Збільшуючи кількість нейронів вихідного шару, можна збільшувати деталізацію результатів процесу кластеризації.

«Нейрони цього функціонують за принципом конкуренції, тобто. внаслідок певної кількості ітерацій активним залишається один нейрон або

нейронний ансамбль (група нейронів, які спрацьовують одночасно). Цей механізм здійснюється за рахунок латеральних зв'язків та називається латеральним гальмуванням» [9]. Оскільки відпрацювання цього механізму потребує значних обчислювальних ресурсів, він організовується штучно, шляхом присвоєння логічної одиниці нейрону з максимальною активністю, а іншим нейрона – логічний нуль.

Зазвичай нейрони розташовуються у вузлах двомірної сітки з прямокутними або шестикутними осередками. При цьому, як було зазначено вище, нейрони також взаємодіють один з одним. Розмір цієї взаємодії визначається відстанню між нейронами на карті. На рисунку 2.4 представлений приклад відстані для шестикутної та чотирикутної мереж.

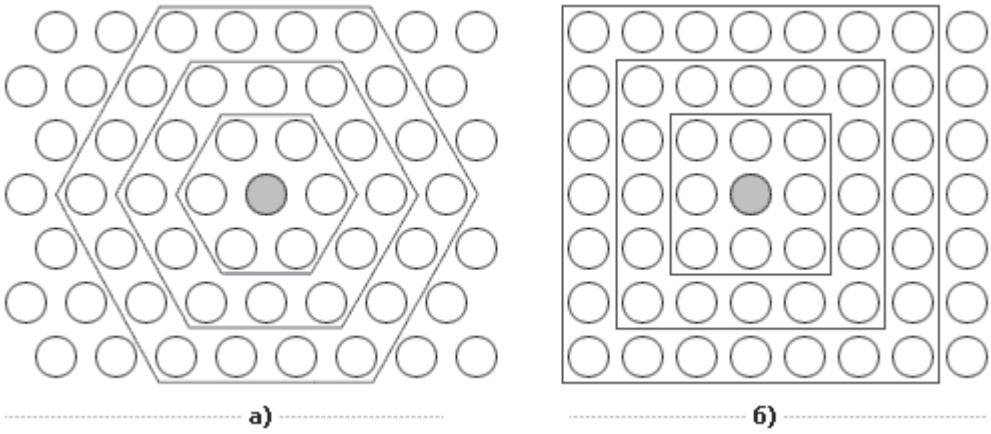


Рисунок 2.4 – Структура мережі Кохонена

Відстань між нейронами на карті для шестикутної (малюнок 2.11а) та чотирикутної (малюнок 2.11 б) сіток. При цьому легко помітити, що для шестикутної сітки відстань між нейронами більше збігається з евклідовою відстанню, ніж для чотирикутної сітки.

Кількість нейронів у сітці визначає ступінь деталізації результату роботи алгоритму, і зрештою від цього залежить точність узагальнюючої здатності карти.

При реалізації алгоритму SOM заздалегідь визначається конфігурація сітки (прямокутна або шестикутна), а також кількість нейронів в мережі.

Деякі джерела рекомендують використовувати максимально можливу кількість нейронів у карті. У цьому початковий радіус навчання значною мірою впливає здатність узагальнення з допомогою отриманої карти. У разі коли кількість вузлів карти перевищує кількість прикладів у навчальній вибірці, то успіх використання алгоритму великою мірою залежить від вибору початкового радіусу навчання. Однак, у випадку, коли розмір картки становить десятки тисяч нейронів, той час, необхідний на навчання карти зазвичай буває занадто велике для вирішення практичних завдань, таким чином є необхідність досягати компромісу при виборі кількості вузлів.

Перед початком навчання картки необхідно проініціалізувати вагові коефіцієнти нейронів. Вдало обраний спосіб ініціалізації може суттєво прискорити навчання, і призвести до отримання якісніших результатів. Існують три способи ініціювання початкових ваг. [9]

– Ініціалізація випадковими значеннями, коли всім вагам даються малі випадкові величини.

– Ініціалізація вхідними даними, коли як початкові значення задаються значення випадково обраних вхідних даних з навчальної вибірки.

– Лінійна ініціалізація, яка полягає у тому, що ваги ініціалізуються значеннями векторів, лінійно впорядкованих уздовж лінійного підпростору, що проходить між двома головними власними векторами вихідного набору даних.

Навчання мережі Кохонена відбувається наступним чином.

Ваговий вектор для кластерного елемента мережі Кохонена є прикладом вхідних сигналів, асоційованих з цим кластером. Під час процесу самоорганізації нейрона, ваговий вектор якого найбільше відповідає вхідному сигналу, вибирається переможцем. Переможець та сусідні з ним елементи змінюють свій ваговий вектор. Наприклад, для лінійного масиву нейронів сусідство радіусом R, розташоване навколо кластерної одиниці J, складається з усіх елементів j таких, що $max(1, J - R) \le j \le min(J + R, m)$.

Сусідства радіусів *R = 2, 1* та 0 показані на рисунку 2.5 для прямокутної мережі. Нейрон, що переміг, позначений символом "#", а решта елементів – "*". Зауважимо, що у прямокутній сітці кожен елемент має вісім найближчих сусідів.

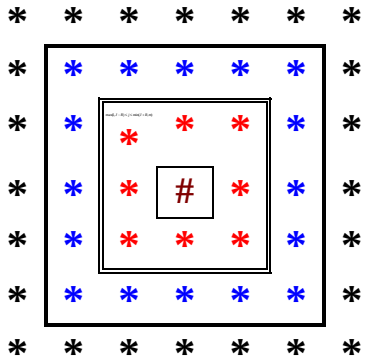


Рисунок 2.5 – Розташування сусідніх нейронів у мережі

Алгоритм навчання може бути описаний таким чином.

1. Ініціалізація вагових коефіцієнтів (можливі варіанти вибору були описані вище). Встановити параметри топологічного сусідства та параметри коефіцієнта навчання.

2. Для кожного вхідного вектора X виконуються кроки 3-5.

3. Для кожного *j* розраховується:

$$d_j = \sum_j (w_{ij} - x_i).$$

4. Серед усіх індексів j знаходиться такий для якого мінімально.

5. Для всіх елементів *j* всередині конкретного сусідства *J* і для всіх *i* проводиться підстроювання ваги за такою формулою:

$$w_{ij}(t+1) = w_{ij}(t) + \alpha(x_i - w_{ij}(t)). \qquad (2.8)$$

6. Змінити коефіцієнт навчання α за обраним правилом.

7. Зменшити радіус топологічного сусідства відповідно до обраного алгоритму.

8. Якщо умови зупинення алгоритму не досягнуто, перейти до пункту 2, інакше закінчення навчання.

Перед подачею на вхід нейромережі, вхідний вектор x нормується, шляхом розташування на гіперсфері одиничного радіусу у просторі ваг. При корекції ваг за правилом (2.8) відбувається поворот вектора терезів у бік вхідного вектора. Поступове зменшення швидкості повороту α дозволяє здійснити статистичне усереднення вхідних векторів, на які реагує даний нейрон.

Геометрично це правило ілюструє рисунком нок 2.6.

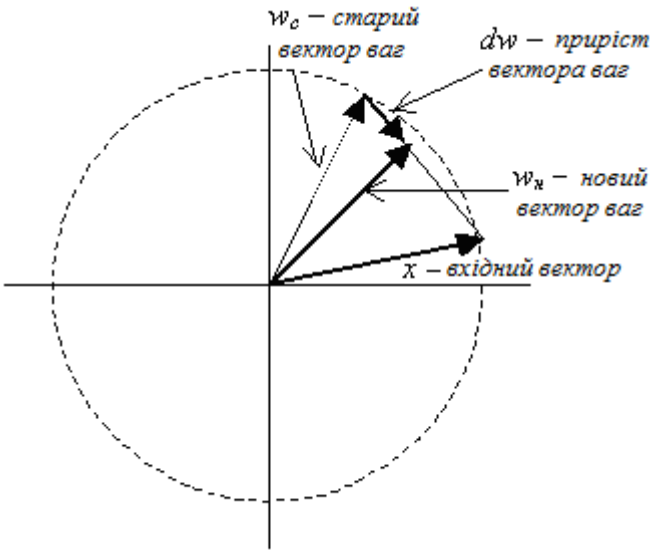


Рисунок 2.6 – Корекція вагів нейрона Кохонена

Коефіцієнт навчання α – змінна величина, що повільно зменшується у часі (або протягом періодів навчання). Для практичних розрахунків цілком прийнятний коефіцієнт зменшується коефіцієнт навчання. Радіус сусідства навколо кластерного елемента також зменшується, як розвивається процес кластеризації. Формування карти відбувається у двох фазах: початкове формування правильного порядку та остаточна збіжність. Друга фаза займає набагато більше часу, ніж перша, і вимагає невеликого значення коефіцієнта навчання.

Початковим ваговим коефіцієнтам можуть надаватися випадкові значення. Якщо є деяка інформація щодо розподілу кластерів, яка підходить для будь-якої приватної проблеми, її можна використовувати для завдання

початкових значень вагових коефіцієнтів з метою підвищення якості навчання. Вагові коефіцієнти встановлюють внаслідок ініціалізації випадкових значень.

На кожному кроці навчання з вихідного набору даних випадково вибирається один із векторів, а потім проводиться пошук найбільш схожого на нього вектора серед усіх векторів ваг нейронів. Далі вибирається нейрон-переможець (в алгоритмі це нейрон для якого було мінімальним), який найбільше схожий на вхідний вектор.

Після того, як знайдено нейрон-переможець, проводиться коригування ваг нейромережі (крок 5 в описаному алгоритмі навчання). При цьому вектор описує нейрон-переможець і вектори сусідів, що його описують, в сітці переміщуються в напрямку вхідного вектора. Цей процес проілюстровано рисунку 2.7 для двовимірного вектора.

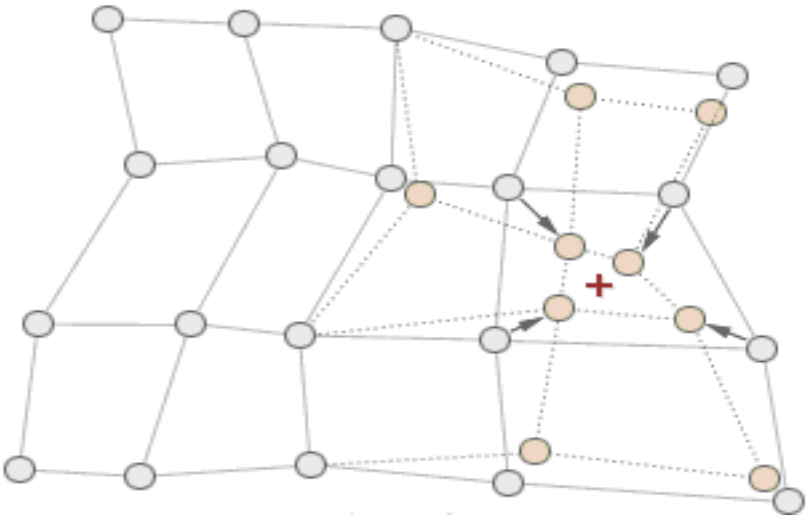


Рисунок 2.7 – Переміщення нейрона переможця.

Підстроювання вагів нейрона переможця та його сусідів. Координати вхідного вектора позначені хрестом, координати вузлів картки після модифікації відображені сірим кольором. Вигляд сітки після модифікації відображається штриховими лініями.

В вище описаному алгоритмі використовується спрощена формула корекції ваги, оригінальна формула зазвичай виглядає як:

$$w_{ij}(t+1) = w_{ij}(t) + h_{ci}(t) \cdot [x(t) - w(t)], \tag{2.9}$$

де *t* означає номер епохи (дискретний час).

При цьому вектор *x(t)* вибирається випадково з навчальної вибірки ітерації *t*. Функція *h(t)* називається функцією сусідства нейронів. Ця функція являє собою функцію, що не зростає, від часу і відстані між нейроном-переможцем і сусідніми нейронами в сітці. Ця функція розбивається на частини, а саме на функцію відстані і функції швидкості навчання в залежності від часу, де визначає положення нейрона в мережі.

Зазвичай застосовується одна з двох функцій відстані:

проста константа

$$h(d,t) = \begin{cases} const, d \le \sigma(t) \\ 0, d > \sigma(t) \end{cases} \tag{2.10}$$

або Гаусова функція

$$h(d,t) = e^{-\frac{d}{2\sigma^2(t)}}. \tag{2.11}$$

Як правило, найкращий результат отримується при використанні Гаусової функції відстані, що є спадною функцією від часу. Часто величину називають радіусом навчання, який вибирається досить великим на початковому етапі навчання та поступово зменшується так, що зрештою навчається один нейрон-переможець. Найчастіше використовується функція, що лінійно спадає в часі.

Розглянемо тепер функцію швидкості навчання. Ця функція також є функцією спадаючою від часу. Найчастіше використовуються два варіанти цієї функції: лінійна і обернено пропорційна часу виду

$$\alpha(t) = \frac{A}{t + B}, \tag{2.12}$$

де *A* і *B* це константи.

Застосування цієї функції призводить до того, що всі вектори з навчальної вибірки роблять приблизно рівний внесок у результат навчання. Навчання складається з двох основних фаз: на початковому етапі вибирається досить велике значення швидкості навчання і радіуса навчання, що дозволяє розташувати вектора нейронів відповідно до розподілу прикладів у вибірці, а потім проводиться точне підстроювання ваги, коли значення параметрів швидкості навчання набагато менше початкових. У разі використання лінійної ініціалізації початковий етап грубого підстроювання може бути опущений.

2.4 Опис шару Гроссберга

Шар Кохонена працює в режимі інтерполяції або акредитації, всі шари пов'язані.

Шар Гроссберга призначений для спільної роботи з шаром, що дає єдину одиницю на виході (як у шару Кохонена в режимі акредитації) або такий набір виходів, що їхня сума дорівнює одиниці. Нейрони шару Гроссберга обчислюють виважену суму входів. Функція активації лінійна. Шар Гроссберга дає на виході лінійну комбінацію своїх векторів ваги, коефіцієнти комбінації задаються входами шару Гроссберга.

Кожен нейрон шару Гроссберга дає на виході один із своїх вагових коефіцієнтів, номер якого збігається з номером активного нейрона Кохонена. Отже, шар Гроссберга перетворює вихід шару Кохонена з кодуванням за номером каналу довільний лінійний код на виході, матриця коду, що породжує, збігається з матрицею вагових коефіцієнтів шару Гроссберга.

Робота мережі у режимі акредитації представлена малюнку 2.8. Показано лише ненульові виходи. Активовано другий нейрон шару Кохонена.

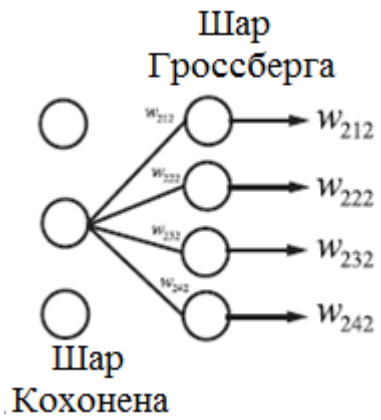


Рисунок 2.8 – Робота нейронів Гроссберга

Нейрон цього шару функціонує класичним варіантом: обчислюється сумарний зважений сигнал на входах і передає його на вихід за допомогою лінійної функції.

Шар Гроссберга навчається щодо просто. Вхідний вектор, що є виходом шару Кохонена, подається на шар нейронів Гроссберга і виходи шару Гроссберга обчислюються природним для більшості мереж чином. Далі, кожна вага коригується лише в тому випадку, якщо він з'єднаний з нейроном Кохонена, що має ненульовий вихід. Величина корекції ваги пропорційна різниці між вагою та необхідним виходом нейрона Гроссберга з яким він з'єднаний. У символьному записі це буде виглядати так:

$$w_{ij\,н} = w_{ij\,с} + \beta \cdot \left(y_j + w_{ij\,с}\right) \cdot x_i, \qquad (2.13)$$

де $w_{ij\,н}$, $w_{ij\,с}$ – ваги звязків до модифікації і після модифікації;

β – швидкість навчання нейромережі, $\beta < 1$;

y_j – вихідне значення нейрона;

x_i – вхідне значення нейрона.

Спочатку β приймається рівним ~0,1 і потім поступово зменшується у процесі навчання. Звідси видно, що ваги шару Гроссберга сходитимуться до середніх величин від бажаних виходів, тоді як ваги шару Кохонена навчаються на середніх значеннях входів. Навчання шару Гроссберга – це навчання з учителем, алгоритм має у своєму розпорядженні бажаний вихід, по якому він навчається. Шар Кохонена, який самоорганізовується і навчається без вчителя, дає виходи в недетермінованих позиціях. Вони відображаються в бажані виходи шару Гроссберга.

За цим правилом (2.13) вектор ваг зв'язків прямує до вихідного вектору, при умові якщо активний вхід.

3. РЕАЛІЗАЦІЯ АВТОМАТИЧНОЇ СИСТЕМИ РОЗПІЗНАВАННЯ МОВЛЕННЄВИХ АУДІО ПОТОКІВ НА ОСНОВІ НЕЙРОМЕРЕЖІ

3.1 Структурно-алгоритмічна організація програми розпізнавання мовленнєвих аудіо потоків

У структурі програми можна виділити основні логічні модулі, кожен з яких повинен виконувати свої завдання. Структурна схема програми представлено рисунку 3.1, а алгоритм її роботи на рисунку 3.2.

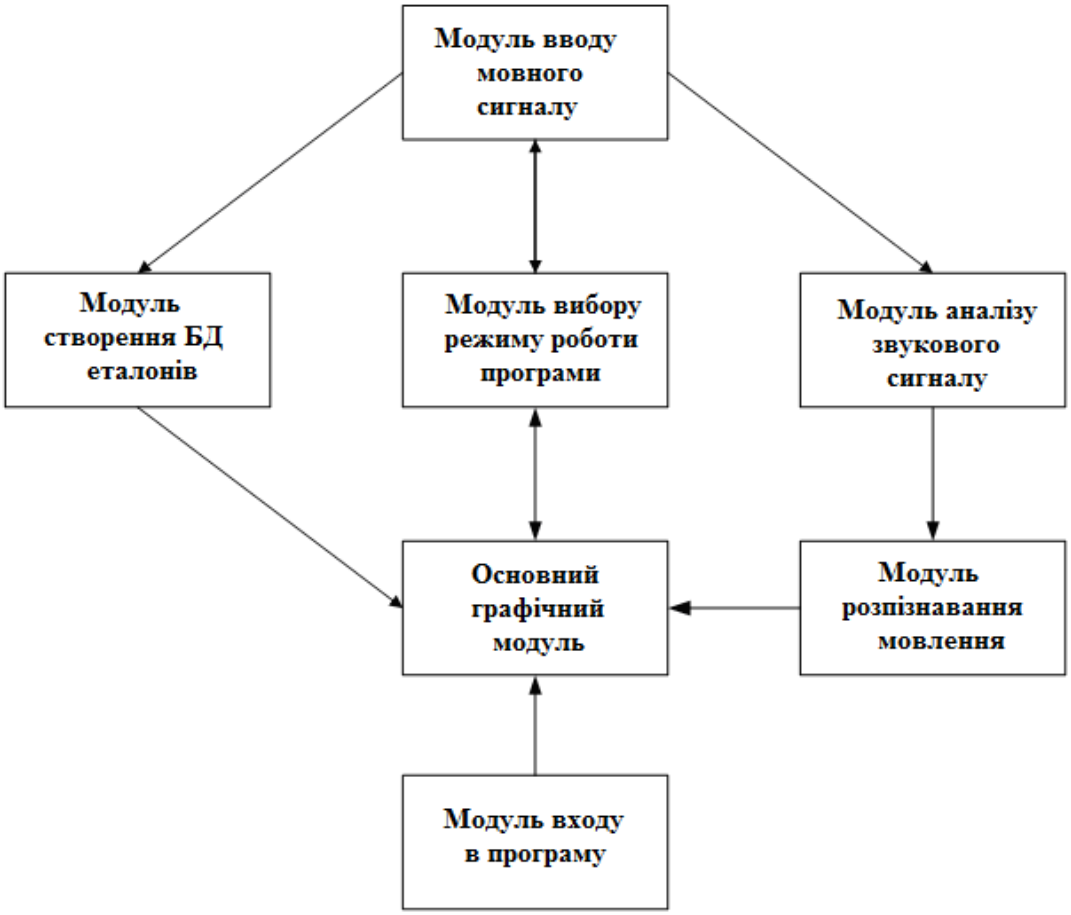


Рисунок 3.1 – Структурна схема програми розпізнавання мовленнєвих аудіо потоків

Звуковий сигнал надходить на перший модуль введення мовного сигналу за допомогою мікрофону або іншим чином, наприклад, через мережу Інтернет.

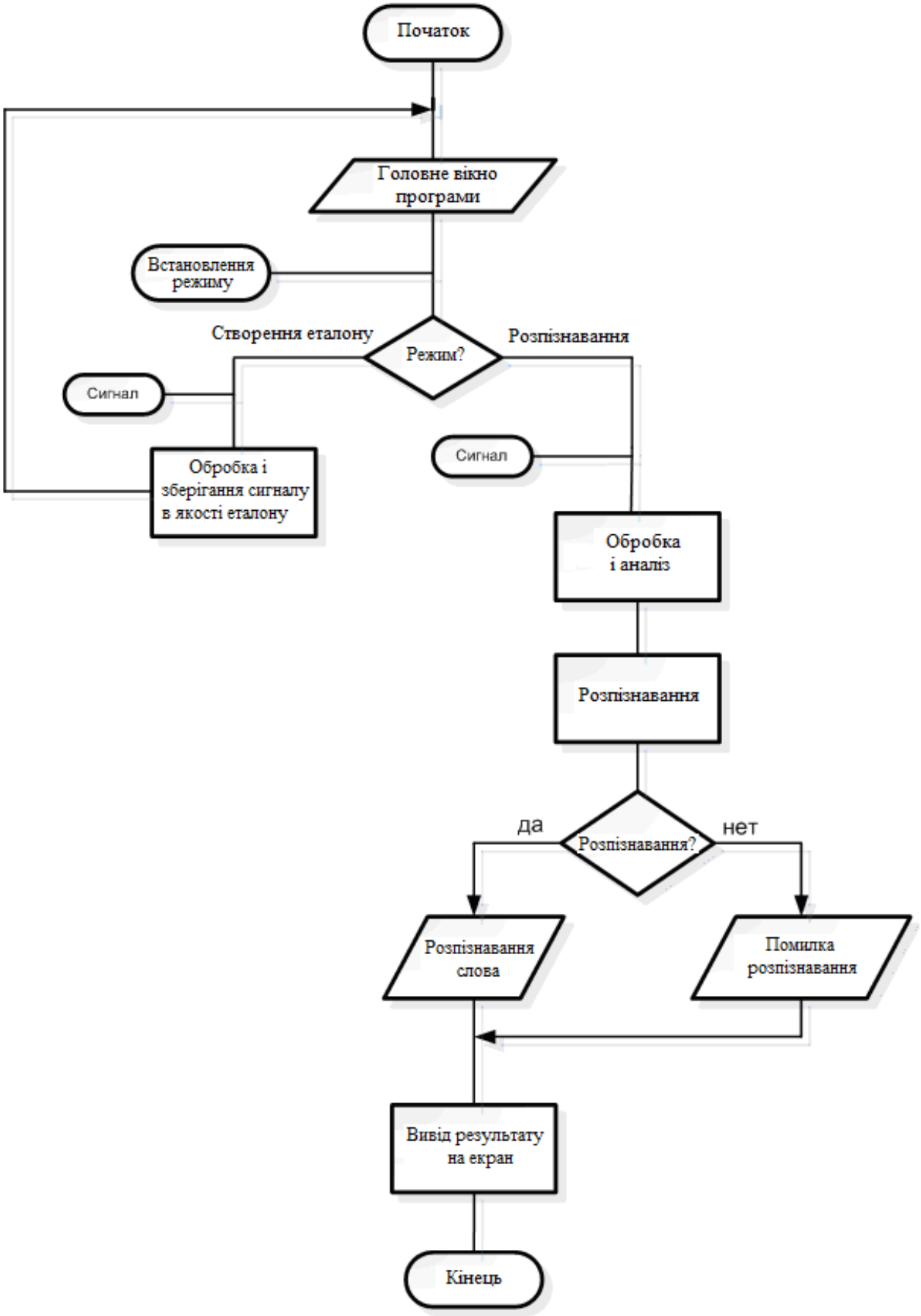


Рисунок 3.2 – Блок-схема алгоритму роботи програми розпізнавання мовленнєвих аудіо потоків

В основному графічному модулі розроблено інтерфейс комунікації користувача з програмою. Користувач при натисканні на відповідні піктограми кнопок на панелі інструментів, або вибору пунктів меню викликає виконання певних функцій інших модулів.

Модуль вибору режиму роботи передбачає обрання користувачем з режим роботи програми.

В модулі аналізу звукового сигналу здійснюється попередня обробка аудіосигналу: фільтрація, представлення у частотному просторі за допомогою перетворення Фур'є та інше, що більш детально описується нижче..

Модуль створення БД еталонів формує на основі аналізує вхідних сигналів базу даних еталонів цифрових мовленнєвих аудіо сигналів.

На заключному етапі модуль розпізнавання мовлення здійснює за допомогою нейронних мереж, тобто аналіз зпівставлення вхідного сигналу з еталонами бази даних.

В аудіо сигналі, що надходить на вхід системи, визначаються межі мовної ділянки – передбачуваної слова – на основі функцій короткочасної енергії сигналу, числа переходів через нуль та кількості точок сталості. Далі виділена мовна команда паралельно аналізується сегментним та цілісним каналами. Сегментний канал заснований на методі ковзного фонетичного аналізу, а цілісний канал – на методі нечіткого зіставлення DTW образів. Ці канали формують незалежні набори слів претендентів, тобто. слів, до кожного з яких з певним коефіцієнтом впевненості може бути віднесена команда, що розпізнається. На останньому рівні схеми, використовуючи набори слів-претендентів та відповідні їм коефіцієнти впевненості, проводиться узгодження наближених рішень сегментного та цілісного каналів і приймається остаточне рішення про команду, що розпізнається.

Аудіо сигнал вимовленого слова представляється як двомірного спектрального тимчасового образу (СТО), одержаного за допомогою віконного перетворення Фур'є (рисунок 3.3, а). СТО дозволяє виділити місце розташування резонансних частот, тобто. локальних викидів, що є

визначальною особливістю мовного сигналу. На цій підставі СТО перетворюється на двійковий вид за допомогою заміни: 1 – на місці локального викиду, 0 – в інших місцях. Отриманий образ є бінарним спектральним тимчасовим образом (БСТО) і використовується як відображення особливостей мовного сигналу (рисунок 3.3, б).

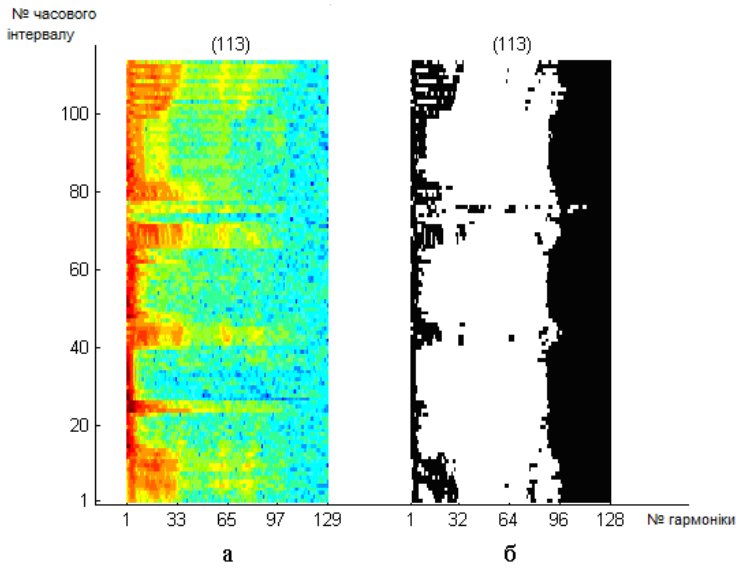


Рисунок 3.3. – Приклад спектрально-часового представлення слова “автоформат”: а) – СТО; б) – БСТО

Як зазначалося у першому розділі нейронні мережі потребують попереднього опрацювання вхідного аудіо сигналу як функції від часу, в представлення його в частотній формі. Якщо такий сигнал заданий в цифровій формі як дискретна послідовність *s(n)*, то частотне представлення отримується за допомогою перетворення Фур'є *S(ω, n)*:

$$S(\omega,n)=\sum_{k=-\infty}^{\infty}s(k)h(n-k)e^{-j\omega k}. \qquad (3.1)$$

В даному виразі використано не класичний вираз перетворення Фур’є, а з використанням вагової функція *h(n)* зсунутої у часі.

Метод аналізу мовних сигналів на основі лінійного передбачення є одним із найефективніших. Такий метод має переваги при оцінці основних

параметрів мовних сигналів, завдяки достатньо високій точності одержуваних оцінок при відносно нескладних обчисленнях. Його використовують для визначення періодів основного тону, спектральних характеристик сигналу, а також для стиснення звукового сигналу, що забезпечує його швидшу передачу та економить ресурси пам'яті при запам'ятовуванні.

Основна ідея цього методу полягає в апроксимації поточного відліку мовного сигналу лінійною комбінацією попередніх дискретних значень сигналу. При цьому, коефіцієнти передбачення однозначно визначається мінімальним значенням середньоквадратичного відхилення дискретних значень мовного сигналу від його передбачуваного значеннями. Ці коефіцієнти використовуються у лінійній комбінації як зважуючі.

Як зазначалося у першому розділі, попереднє опрацювання мовних сигналів має за мету створення цифрової моделі сигналу, що відсікає всі непотрібні параметри для зменшення об'єму пам'яті, зручнішому представленні образу сигналу для його подальшої обробки нейромережею та компактнішого представленн. Крім того, за допомогою попередньої цифрової обробки здійснюється фільтрація корисного сигналу від шумів та завад та визначенні чи взагалі в сигналі присутнє мовлення, а не сторонній шум. Також можливе розділення аудіо потоку на корисні – мовленнєві інтервали та паузи чи інші проміжки між словами, фразами та реченнями.

Для аналізу аудіо сигналу мовлення в часовій області найбільш відомими є метод, розроблений Л. Рабінером і Р. Шафером в [3]. Його суть полягає на визначенні середнього значення сигналу та короткочасної функції середньої кількості переходів через нуль. Вище зазначалося, що амплітуда мовленнєвого аудіо сигналу сильно змінюється на короткому інтервалі часу. Тому такий аудіо сигнал можна достатньо точно представити за допомогою короткочасної дискретної функції сигналу. Тоді енергія сигналу, заданого в (3.1) буде визначатися за виразом:

$$E_n = \sum_{m=-\infty}^{\infty} [x(m)w(n-m)]^2$$

Цей вираз може бути записаний у іншому вигляді, як:

$$E_n = \sum_{m=-\infty}^{\infty} x^2(m)h(n-m), \qquad (3.2)$$

де $h(n) = w^2(n)$.

Характер функції вагових коефіцієнтів $h(n)$ і ширина вікна, що задається дискретним зсувом будуть кардинальним чином впливати на функціє енергії сигналу.

Для дослідження впливу ширини вікна $(n-m)$ на функцію енергії сигналу(3.2), приймемо, що $h(n)$ в (3.1) є досить тривалою і зсталою амплітудою. Тоді значення енергії E_n в часі буде змінюватися дуже мало. Таке вікно працює аналогічно фільтру нижніх частот із вузькою смугою пропускання. Проте смуга пропускання цього вікна не повинна бути занадто вузькою, щоб вихідний сигнал перетворився в постійний. Тоді для адекватного відображення швидких змін амплітуди аудіо сигналу, потрібно використовувати достатньо вузьке вікно, але враховувати, що занадто мала ширина імпульсної функції $h(n-m)$ призведе до недостатнього усереднення значень аудіо сигналу, і як наслідок – недостатнього згладження функції енергії сигналу (3.2). Таким чином, можна зробити висновки про вплив ширини часового вікна на адекватність визначення ковзного математичного сподівання по часу короткочасного сигналу чи математичного сподівання енергії сигналу:

якщо дискретна ширина вікна $N = n - m$ мала, рівна або менша періоду основного тону мовленнєвого аудіо сигналу, то енергія цього сигналу E_n буде дуже швидко змінюватися, відносно до загальної структури мовленнєвого сигналу;

якщо занадто велике, тобто більша тривалості декількох періодів основного тону, то енергія E_n буде змінюватися занадто повільно і також не буде адекватно описувати особливості мовленнєвого аудіо сигналу в часі.

Це означає, що немає єдиного значення, яке повною мірою

задовольняло б переліченим вимогам, оскільки період основного тону може змінюватися в діапазоні від $N=10$ дискретних значень при частоті дискретизації $\Delta t=10кГц$ – для високих дитячих і жіночих голосів, і до $N=250$ дискретних відліків – для дуже низьких чоловічих голосів. Задамо значення ширини вікна N рядом значень 100, 200, 300 при частоті дискретизації 8 кГц.

Обчисливши значення енергії сигналу E_n можна визначити, чи даний фрагмент звукового запису містить мовленнєві складові, чи це є сигналом іншого походження. Мовленнєві фрагменти сигналу мають енергію значно більшу ніж фрагменти з їх відсутністю. Таким чином, порівнявши E_n з деяким пороговим значенням можна відразу відсіяти інтервали, що не потребують подальшого розпізнавання, оскільки не є мовленнєвими.

Також для аналізу мовленнєвих аудіо сигналів використовується бінарне квантування [8], що дозволяє визначити ряд їх параметрів. Формалізовану математичну модель бінарного квантування можна представити виразом:

$$S(t)=A(t)\cdot e^{j\Psi(t)}, \tag{3.3}$$

де $A(t)$ – амплітудна складова мовленнєвого сигналу, $\Psi(t)$ – фазова характеристика мовленнєвого сигналу.

Амплітудна характеристика $A(t)$ має інваріантний характер в часі для однієї і тієї самої мовленнєвої одиниці, висловленою з різною гучнісю чи інтонацією, тому є не достатньо інформативнив параметром для оцінки мовленнєвого повідомлення. А фазова складова – не чутлива до інваріантності висловлювань і підходить в якості ознаки. Її можна представии у вигляді розкладу в ряд Тейлора. для більшої зручності її обробки:

$$\Psi(t)=\frac{\Psi^{(0)}(t_0)}{0!}t^0+\frac{\Psi^{(1)}(t_0)}{1!}t^1+\frac{\Psi^{(2)}(t_0)}{2!}t^2+\frac{\Psi^{(3)}(t_0)}{3!}t^3+\ldots. \tag{3.4}$$

Вираз (3.4) можна переписати у вигляді:

$$\Psi(t) = \mu_0 + \mu_1 t + \frac{\mu_2 t^2}{2} + \frac{\mu_3 t^3}{6} + \ldots. \qquad (3.5)$$

де коефіцієнти $\mu_i = \frac{\Psi^{(i)}(t_0)}{i!}$.

Для спрощення обчислень до уваги приймаються тільки три перші члени ряду. Оскільки перший член ряду є константою не змінною у часі, фізичний зміст якого полягає в тому, що він виражає початкову фазу мовленнєвого аудіо сигналу і є неінформативним, то він приймається рівним нулю. Тоді спрощене представлення фазової функція сигналу прийме вигляд:

$$\Psi(t) = \mu_1 t + 0{,}5\mu_2 t^2, \qquad (3.6)$$

де μ_1 – коефіцієнт розкладу, що є середньою частотою мовного сигналу,

μ_2 – коефіцієнт розклду, що є зміною (девіацією) частоти мовного сигналу.

Якщо перейти від аналогової форми до цифрового представлення, отримаємо:

$$\Psi(i \cdot \Delta t) = \mu_1 \cdot (i \cdot \Delta t) + 0{,}5 \cdot \mu_2 \cdot (i \cdot \Delta t)^2, \qquad (3.8)$$

де i – номер поточного відліку в дискретизованій послідовності, Δt – крок дискретизації.

Функції μ_1 та μ_2 є параметрами, для представлення опису мовленнєвого сигналу. Обчисливши першу і другу різницю можна з деякою похибкою грубо визначити значення μ_1, що є оціночним значенням частоти сигналу та μ_2, що виражає швидкість зміни кількості переходів через нуль мовленнєвого сигналу відповідно. Вирази для обчислення першої і другої різниці фазової функції сигналу мають такий вигляд [4]:

$$\Delta\Psi(i\cdot\Delta t)=\Psi(i\cdot\Delta t)-\Psi((i-L)\cdot\Delta t)=\mu_1\cdot L\cdot\Delta t+\mu_2\cdot i\cdot L\cdot\Delta t^2-0{,}5\cdot\mu_2\cdot(L\cdot\Delta t)^2,$$

$$\Delta^2\Psi(i\cdot\Delta t)=\Delta\Psi(i\cdot\Delta t)-\Delta\Psi((i-L)\cdot\Delta t)=\mu_2\cdot(L\cdot\Delta t)^2, \qquad (3.10)$$

де L – ширина, виражена в кількості відліків, "ковзного" вікна.

Тоді з (3.9) частоту мовного сигналу μ_1 та зміну частоти μ_2, отримаємо у вигляді:

$$\mu_2=\Delta^2\Psi(i\cdot\Delta t)/T^2$$

$$\mu_1=(\Delta\Psi(i\cdot\Delta t)-\mu_2\cdot i\cdot T\cdot\Delta t+0{,}5\cdot\mu_2\cdot T^2)/T,$$

де $T=L\cdot\Delta t$ – ширина "ковзного" вікна.

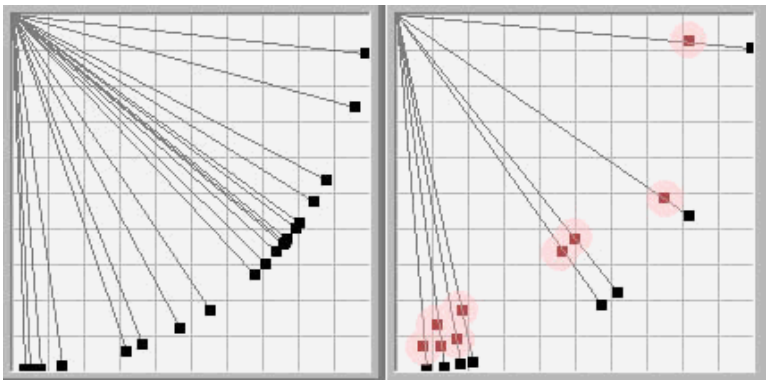
3.2 Навчання нейромережевої моделі розпізнавання мовлення

Навчання нейромережі відбувається поетапно. Спочатку системі надаються тільки зразки звуків, при цьому у вхідному шарі формуються нейронні ансамблі, ядрами яких є надані зразки. Потім надаються звуки та відповідні символи алфавіту та відбівається створення асоціативних зв’язків нейронів вхідного рівня із нейронами символьного шару. Останній етап полягає у навчанні нейромережі синтезу мовлення. Це навчання відбуається без завантаження додаткових зразків чи даних за рахунок використання інформації, отриманої на попередніх етапах. Для навчання нейромережі використовується стохастичне навчання, суть якого полягає у випадкових змінах нейронів ефекторного шару, що генерує звук. Цей звук надходить на вхідий шар, розпізнається і результат порівнюється з тим символом, для якого було згенеровано звук. При правільному розпізнаванні, результат запам’ятовується. Цей процес для всіх звуків мовлення, доки не буде досягнуто правильної генерації і розпізнавання всіх звуків.

1. Вибір значень початкових ваг.

Так як наприкінці навчання вектори ваг будуть відображатися на одиничному колі, то на початку їх також бажано віднормувати до одиниці.

Тому зазвичай вектори терезів вибираються випадковим чином на колі одиничного радіусу, як це показано на рисунку 3.4.



а) б)

Рисунок 3.4 – Моделювання шару Кохонена

а) початкові значення ваг; б) ваги після навчання

2. Вибір швидкості навчання

Очевидно, що для кожного нейрона існує безліч вхідних сигналів, які могли б його активувати, і його вектор зв'язків постійно змінювався б. Якщо коефіцієнт навчання $\alpha < 1$, тоді на кожен вхідний сигнал вектор зв'язків реагує мало. Завдяки постійному зменшенню α у процесі навчання, на його завершенні сформуються статистичні усереднення подібних вхідних сигналів. Тому вводяться швидкості навчання у всіх інших навчальних правилах.

Швидкість навчання визначається порядком подання зразків. Таким чином, головний критерій вибору швидкостей навчання – незначна зміна зв'язків у межах усієї навчальної вибірки. Але оскільки час навчання обернено пропорційно швидкості навчання, необхідно шукати компроміс.

3. Запам'ятовування елементів, що рідко зустрічаються.

Описаний вище алгоритм навчання хороший для сигналів, що часто повторюються. Якщо ж сигнал зустрічається рідко на тлі всієї навчальної вибірки, він просто не запам'ятається. У такому разі необхідно залучити механізм уваги [5]. З появою невідомого нейромережі зразка швидкість

навчання багаторазово зростає і рідкісний елемент запам'ятовується в нейромережі. У системі, що розробляється, навчальна вибірка будується штучно, тому такої проблеми не виникає, і механізм уваги не реалізований. Необхідність механізму уваги з'являється під час навчання у природних умовах, коли навчальна вибірка заздалегідь не передбачувана.

4. Використання всіх нейронів

Якщо ваговий вектор виявиться далеко від області вхідних сигналів, він ніколи не дасть найкращої відповідності, завжди матиме нульовий вихід, отже, не коригуватиметься і виявиться марним. Решту нейронів може не вистачити для поділу вхідного простору сигналів на класи. Для вирішення цієї проблеми пропонується багато алгоритмів ([8,9]), наприклад, правило «бажання працювати»: якщо якийсь нейрон довго не знаходиться в активному стані, він підвищує ваги зв'язків до тих пір, поки не стане активним і не почне навчатися. Цей метод дозволяє також вирішити проблему тонкої класифікації: якщо утворюється група вхідних сигналів, розташованих близько один до одного, з цією групою асоціюється і велика кількість нейронів Кохонена, які розбивають її на класи.

Правило «бажання працювати» записується у такій формі:

$$w_н = w_c + w_c \cdot \alpha_1(1-a), \tag{3.11}$$

де $w_н$ – нове значення ваги;

w_c – старе значення;

α_1 – швидкість модифікації;

a – активність нейрона.

Чим менша активність нейрона, тим більше збільшуються ваги зв'язків.

Вибір коефіцієнта α_1 визначається з таких міркуваннь: постійне зростання ваг нейронів за правилом (3,11) компенсується правилом (2.8) (активні нейрони намагаються знову повернутися на гіперсферу одиничного

радіусу), причому за одну ітерацію нейромережі збільшать свою вагу практично всі нейрони, а зменшить лише один активний нейрон чи нейронний ансамбль. У зв'язку з цим коефіцієнт α_1 (2.13) необхідно вибирати значно менше коефіцієнта α в (2.8), враховуючи при цьому кількість нейронів у шарі.

Функціональна схема роботи двоканальної системи розпізнавання мовлення представлена рисунку 3.5.

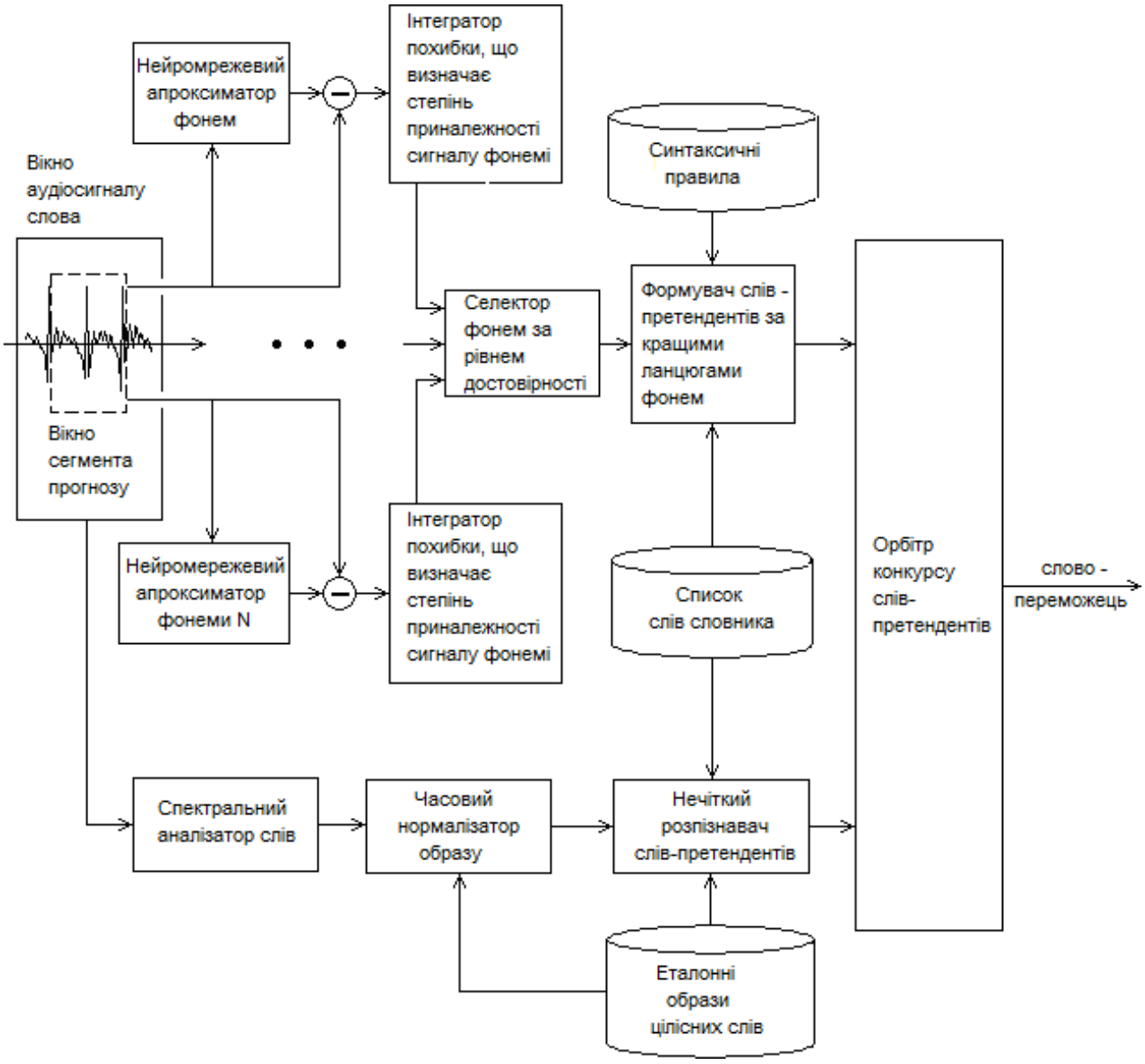


Рисунок 3.5 – Функціональна схема моделі розпізнавання мовлення

3.3 Моделювання та навчання мережі Кохонена у системі Trajan 2.1

Система Trajan 2.1 (Registered Shareware Version) розроблена фірмою Trajan-Software (США). Мережі Кохонена у цій системі реалізується шляхом описаним нижче.

Мережі Кохонена здійснюють режим навчання без «премій» (самонавчання) та мають два шари: вхідний шар та вихідний – шар топологічної карти. Як зазначалося вище, вихідний шар часто представляється двомірною решіткою (має два виміри) і складається з радіальних елементів. Система Trajan підтримує мережі з одновимірним вихідним шаром. У мережі Кохонена помилкою вважається відстань між ваговим вектором нейрона-переможця і вхідним вектором.

Таким чином, спочатку створюється мережа Кохонена у вікні «Створення ІНМ» (Network Creation, рисунок 3.6), яке викликається командою меню «Файл/Створити/Мережу» (File/New/Network)».

Для представлення двовимірного вихідного шару в системі Trajan задаються число нейронів у шарі («Units») та ширина шару («Width»): система визначає висоту шару як відношення загального числа нейронів вихідного шару до його ширини.

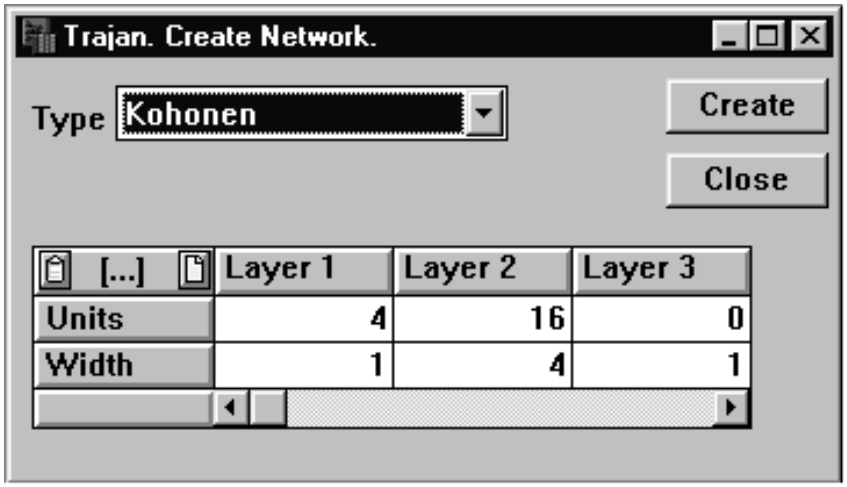


Рисунок 3.6 – Вікно створення мережі Кохонена

Для будь-якого шару будь-якого типу мережі можна задавати параметр width, який є суттєвим для мережі Кохонена.

Потім задаються параметри шарів. Наприклад, для мережі, представленої на рисунку 3.7, задаються такі значення: перший шар – «Units – 4», «Width – 1»; другий шар – «Units – 16», «Width – 4». Після натискання кнопки «Створити / Create на екрані з'являється структура створеної мережі Кохонена, як це показано на рисунку.

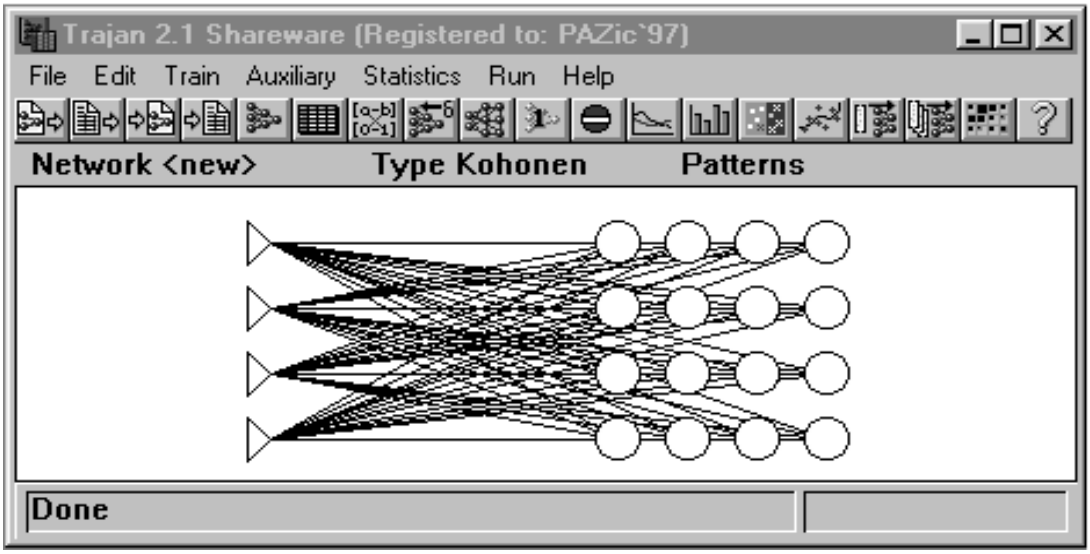


Рисунок 3.7 – Структура створеної мережі Кохонена

Для навчання мереж Кохонена в системі Trajan передбачені:

- вікно «Частоти перемог (Win Frequencies)» для ілюстрації, де у мережі сформувалися кластери;

- вікно «Топологічна карта» для показу, який образ віднесений до якого кластера. Це вікно допоможе заздалегідь задати імена нейронам та образам.

Слід завантажити послідовність вхідних образів, відкрити меню «Файл/Відкрити/Зразок (File/Open/Pattern)» і вибрати необхідний файл у вікні.

Вікно «Навчання Кохонена (Kohonen Training)» (рисунок 3.8) (команда меню «Навчання/Kohonen (Train/Kohonen))» включає початкові та кінцеві параметри для швидкості навчання (коефіцієнта корекції) та для розміру

області близькості (сусідства). Зазвичай навчання мережі Кохонена розбивають на дві частини – фаза прикидки та фаза підстроювання.

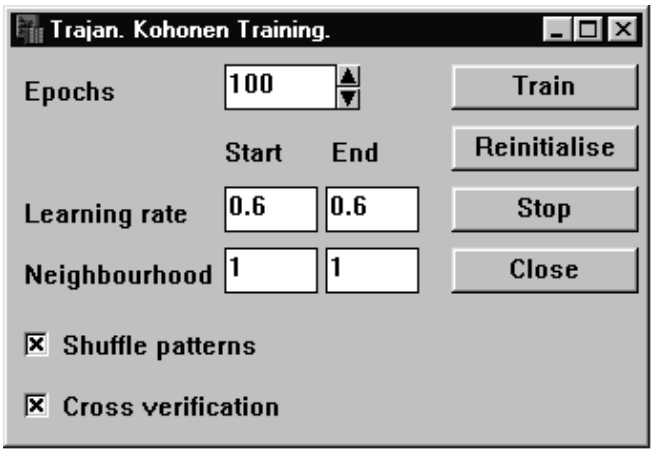


Рисунок 3.8 – Навчання мережі Кохонена

"Швидкість навчання (Коефіцієнт корекції) / Learning Rate" в алгоритмі Кохонена лінійно змінюється від першого циклу навчання до останнього. Зазвичай алгоритм навчання запускають у дві стадії: на першій стадії використовують високу швидкість навчання (наприклад, від 0.9 до 0.1), великий розмір околиці (наприклад, від 2 до 1) та невелику кількість циклів (наприклад, 100), а на другій – незмінні швидкість навчання (наприклад, 0.01), невеликий розмір околиці (наприклад, 0) та велика кількість циклів (наприклад, 10000).

«Навколо» визначає число сусідніх нейронів, що розглядаються, навколо нейрона-переможця, вагові коефіцієнти яких коригуються.

"Реініціалізація / Reinitialise". Під час роботи алгоритму Кохонена натискання кнопки «Реініціалізація/Reinitialise» змінює лише вихідний шар радіальних елементів.

При першому запуску можна зменшити «Швидкість навчання/Learning Rate» від початкового значення 0.5 до кінцевого 0.1 та залишити постійним розмір «Околиці/Neighbourhood» – 1. При другому запуску можна встановити постійну «Швидкість навчання/Learning Rate» 0.1 за розміром «Навколо Neighbourhood» дорівнює 0.

На рисуннку 3.9 наведено діаграму помилок окремих образів, але в рисунку 3.10 – зміна середньої квадратичної помилки за всіма образами.

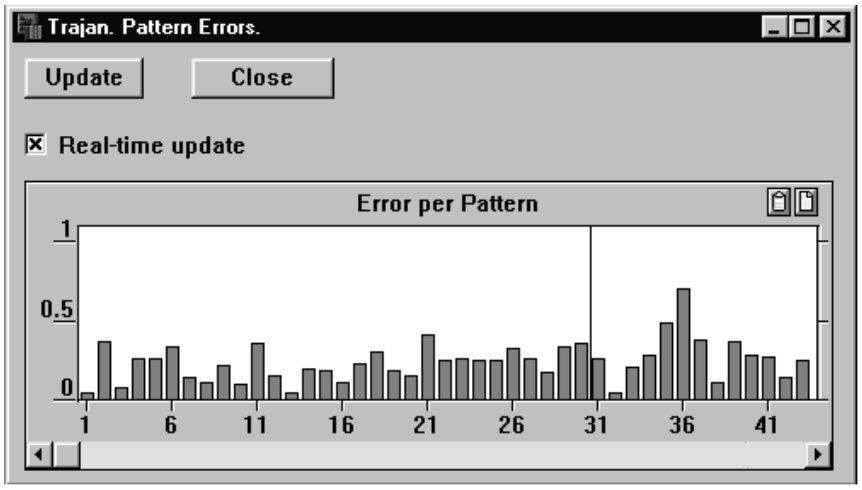


Рисунок 3.9 – Діаграма помилок на кожний образ

Графік помилки навчання. Середня квадратична помилка мережі з усіх образів зображується на графіку помилки, як показано рисунку 3.10.

У мережах Кохонена помилка обчислюється як відстані між вхідним вектором і ваговим вектором нейрона – “переможця” вихідного шару. На графіці показується середня квадратична помилка мережі з усіх образів.

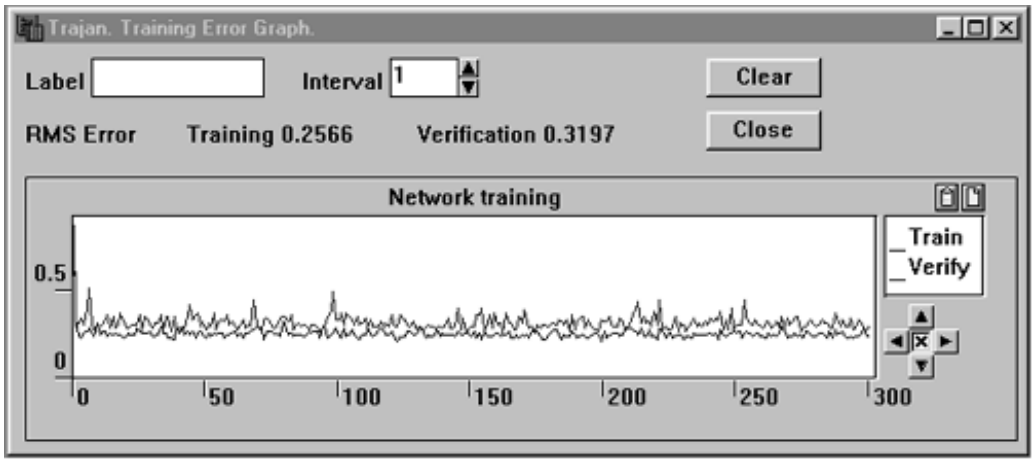
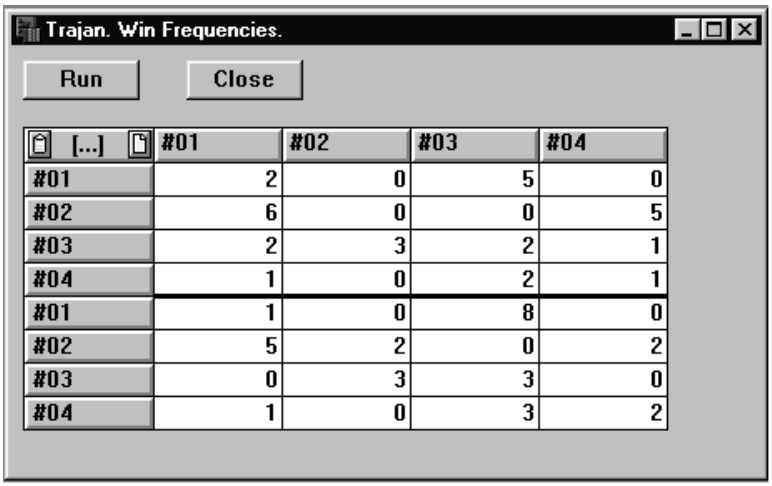


Рисунок 3.10 – Графік помилки навчання

Після навчання мережі можна проаналізувати сформовані кластери та їх зміст.

Налаштування нейромережі Кохонена здійснюється за допомогою вікна "Частоти перемог (Win Frequencies)" і "Топологічна карта (Topological Map)" служать для аналізу мереж Кохонена та здійснення кластеризації. Вікно "Частоти перемог (Win Frequencies)" (рисунок 3.11) викликається командою меню "Статистика/Частоти перемог (Statistics/Win Frequencies)". Система проганяє при цьому всі вхідні образи і підраховує, скільки разів кожен нейрон топологічного шару виграє (тобто знаходиться найближче до образа, що перевіряється). Висока кількість перемог показує центри кластерів на топологічній карті. Нейрони з нульовими частотами перемог не використовуються, їх наявність зазвичай показує, що навчання було не дуже успішним (оскільки мережа використовує не всі надані їй ресурси). Однак у ряді випадків через невелику кількість вхідних образів допустима наявність невикористаних нейронів.



[...]	#01	#02	#03	#04
#01	2	0	5	0
#02	6	0	0	5
#03	2	3	2	1
#04	1	0	2	1
#01	1	0	8	0
#02	5	2	0	2
#03	0	3	3	0
#04	1	0	3	2

Рисунок 3.11 – Вікно результатів частоти перемог

Частоти перемог (Win Frequencies) показуються окремо для навчальних та перевірочних образів (розділені товстою горизонтальною лінією). Якщо розташування кластерів значно відрізняється у цих двох половинах, це означає, що мережа не навчилася правильно узагальнювати дані.

Як тільки розподіл центрів кластерів визначено, можна відкрити вікно «Topological Map» та переглянути мережу з метою ідентифікації кластерів, як

показано на рисунку 3.12. "Топологічна карта" (Topological Map) графічно відображає вихідний шар у двомірному просторі. Для кожного нейрона топологічного шару показується його близькість до поточного образу за допомогою чорного квадрата (що більше квадрат, тим ближче), і нейрон-переможець обведений тонким прямокутником.

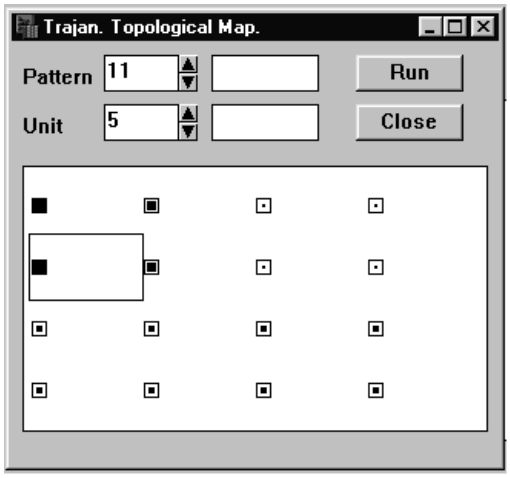


Рисунок 3.12 – Топологічна карта

Під час перевірки кількох образів (шляхом натискання стрілки вгору праворуч від поля «Зразок (Pattern))» можна встановити, що близькі образи об'єднані групи, а близькі нейрони розташовані поруч друг з одним. На цьому етапі можна почати присвоювати нейронам осмислені імена для того, щоб показати їх приналежність до кластера. У нашому прикладі перші десять образів належали до вигляду Setosa.

При цьому слід виконати (Run) для першого образу і вказати ім'я нейрона-переможця: ввести назву Setosa в поле імені елемента «Модуля» (справа від поля номера елемента) і натискання RETURN. Топологічна карта (Topological Map) автоматично оновиться та відобразить нове ім'я. Потім слід проганяти решту дев'яти образів (натискаючи стрілку вгору праворуч від поля зразка) і назвати нейрони-“переможці” аналогічно.

Зауваження. Для виключення набору на клавіатурі того самого імені необхідно скопіювати його в буфер (виділити і натиснути CTRL+INSERT), а

потім вставити його в потрібне місце шляхом натискання SHIFT+INSERT. Можна також не натискати RETURN після кожного імені. При натисканні стрілки нагору топологічна карта автоматично оновлюється.

Потрібно дати назви всім нейронам. Образи 11-20 відносяться до виду Versicolour, образи 21-30 до виду Virginica. При виявленні нейронів, які виграли в обох цих видах, слід зазначити їх як Dubious.

Після перевірки всіх образів послідовності потрібно відзначити решту (невідомих) нейронів як «Невикористані» або запустити знову всі образи і подивитися, якому типу образів невикористані нейрони відповідають найбільш і дати їм відповідну назву. Щойно всі нейрони пойменовані, можна побачити, як мережа Кохонена класифікувала перевірочні образи.

При використанні мережі Кохонена за відсутності інформації про те, які повинні бути кластери (що характерно для застосування мережі Кохонена), необхідно надати отриманим кластерам символічні імена, а потім проаналізувати вхідні дані та визначити типи кластерів. Для цього в системі у вікні топологічної карти (Topological Map) можна давати імена образам, як показано на рисунку 3.13.

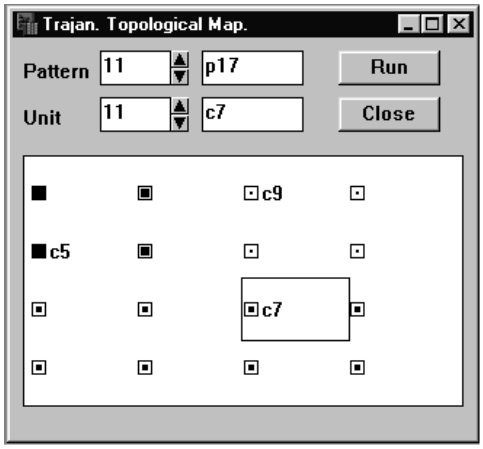


Рисунок 3.13 – Топологічна карта з назвами елементів.

Таким чином, за допомогою двохрівневої нейронної мережі реалізовано розпізнавання мовленєвих аудіо сигналів. Очевидно, що якість розпізнавання буде коливатися в залежності від багатьох зовнішніх чинників.

ВИСНОВКИ

В роботі розроблено структурну схему та змодельовано роботу в програмному середовищі Trajan 2.1 системи автоматичного розпізнавання мовної аудіо інформації з урахуванням прихованих Марківських моделей (СММ) на базі дворівневих нейронних мереж Гроссберга і Кохонена. В результаті проведених досліджень було отримано наступні результати

1. Проаналізовано сучасний стан розвитку систем автоматичного розпізнавання мовленнєвих аудіо потоків (РМАП).

2. На основі проведеного аналізу визначено основні трудності у вирішенні задачі РМАП та на основі сучасного досвіду зроблено висновок, що для вирішення поставленої задачі найкраще підходять нейронні мережі.

3. Проаналізовано структури і функціонування нейронних мереж та обґрунтовано обрання дворівневої НМ на базі мереж Гроссберга і Кохонена

4. Розроблено структурну та функціональну схему системи РМАП на основі дворівневої нейронної мережі.

5. Розроблено структурну схему та алгоритм роботи програми розпізнавання мовленнєвих аудіо потоків.

6. Представлено математичний апарат для початкової обробки вхідних аудіо сигналів на основі дискретного перетворення Фур'є.

7. Здійсненно моделювання та навчання мережі Кохонена у системі Trajan 2.1 для автоматичного РМАП.

8. Розроблено програмне забезпечення для реалізації методу РМАП на основі дворівневої нейронної мережі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Електронний ресурс. Использование распознавания речи в Windows Режим доступу: https://support.microsoft.com/ru-ru/windows/%D0%B8%D1%81%D0%BF%D0%BE%D0%BB%D1%8C%D0%B7%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5-%D1%80%D0%B0%D1%81%D0%BF%D0%BE%D0%B7%D0%BD%D0%B0%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F-%D1%80%D0%B5%D1%87%D0%B8-%D0%B2-windows-83ff75bd-63eb-0b6c-18d4-6fae94050571

2. Електронний ресурс. Розпізнавання мовлення від Google Cloud: навіщо використовувати цей сервіс. Режим доступу: https://cloudfresh.com/ua/cloud-blog/google-speech-to-text-navishcho-vykorystovuvaty/

3. Електронний ресурс. Research Article “An Overview of Speech Recognition Using HMM” Ms. Rupali S Chavan , Dr. Ganesh. S Sable Режим доступу: https://www.researchgate.net/publication/335714660_An_Overview_of_Speech_Recognition_Using_HMM

4. Електронний ресурс. Chat GPT4 и возможности более точного распознавания речи. Режим доступу: https://ts2.space/ru/chat-gpt4-%D0%B8-%D0%B2%D0%BE%D0%B7%D0%BC%D0%BE%D0%B6%D0%BD%D0%BE%D1%81%D1%82%D0%B8-%D0%B1%D0%BE%D0%BB%D0%B5%D0%B5-%D1%82%D0%BE%D1%87%D0%BD%D0%BE%D0%B3%D0%BE-%D1%80%D0%B0%D1%81%D0%BF%D0%BE%D0%B7%D0%BD/#gsc.tab=0

5. Рабинер Л.Р. Шафер Р.В. Цифровая обработка речевых сигналов, М.: Радио и связь, 1981 ., 258 с.

6 Основи та методи цифрової обробки сигналів: від теорії до практики: навч. посібник / уклад. : Ю.О. Ушенко, М.С. Гавриляк, М.В. Талах, В.В. Дворжак. – Чернівці : Чернівецький нац. ун-т ім. Ю. Федьковича, 2021. 308 с.

7. Наконечний А. Й. та ін.. Цифрова обробка сигналів. Навчальний посібник / А. Й. Наконечний, Р. А. Наконечний, В. А. Павлиш. Львів: Видавництво Львівської політехніки, 2010. 368 с.

8. Ваврук Є. Я. Алгоритми та засоби обробки сигналів : навч. посібн. / Ваврук Є., Лашко О., Попович Р. – Львів : СПОЛОМ, 2021. – 240 с.

9. В. Рябенький, Л. Солобуто Моделювання пристроїв обробки цифрових сигналів. КОНДОР, 2021. –352с.

10. Read more at Cloudfresh: https://cloudfresh.com/ua/cloud-blog/google-speech-to-text-navishcho-vykorystovuvaty/

11. Добровська Л. М. Д56 Теорія та практика нейронних мереж : навч. посіб. / Л. М. Добровська, І. А. Добровська. – К. : НТУУ «КПІ» Вид-во «Політехніка», 2015. – 396 с.

12. Nils J. Nilsson. THE QUEST FOR ARTIFICIAL INTELLIGENCE A HISTORY OF IDEAS AND ACHIEVEMENTS/ Web Version Print version published by Cambridge University Press:
http://www.cambridge.org/us/0521122937

13. Thomas G. Dietterich and Xinlong Bao, “Integrating Multiple Learning Components through Markov Logic,” Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, pp. 622–627, 2008. A preprint is available at http://web.engr.oregonstate.edu/tgd/publications/aaai08-ilr.pdf .

14. Nils J. Nilsson. THE QUEST FOR ARTIFICIAL INTELLIGENCE A HISTORY OF IDEAS AND ACHIEVEMENTS/ Web Version Print version published by Cambridge University Press:
http://www.cambridge.org/us/0521122937

15. Нейронна мережа (Neural network) [Електронний ресурс] – Режим доступу: https://wiki.loginom.ru/articles/neural-network.html (дата звернення: 02.09.2023).

16. Hlavcheva, D., та V. Yaloveha. "КАПСУЛЬНІ НЕЙРОННІ МЕРЕЖІ". Системи управління, навігації та зв’язку. Збірник наукових праць 5, № 51 (30 жовтня 2018): 132–35.

http://dx.doi.org/10.26906/sunz.2018.5.132.

17. Федоряка, М., та К. Мелкумян. "Гібридний метод обробки зображень на конволюційних нейронних мережах". Адаптивні системи автоматичного управління 1, № 38 (31 травня 2021): 72–76. http://dx.doi.org/10.20535/1560-8956.38.2021.233198 .

18. Штучні нейронні мережі [Електронний ресурс] – Режим доступу: https://www.it.ua/ru/knowledge-base/technology-innovation/iskusstvennyenejronnye-seti-ins (дата звернення: 02.09.2021).

19. Pogrebnyak, S. V., та O. O. Vodka. "Моделювання механічної поведінки еластомірних матеріалів за допомогою штучної нейронної мережі". Scientific Bulletin of UNFU 28, № 11 (27 грудня 2018): 130–34. http://dx.doi.org/10.15421/40281123 .

20. Ковнер, А. А. "Нейронні мережі в робототехніці". Thesis, Сумський державний університет, 2018. http://essuir.sumdu.edu.ua/handle/123456789/67036.

21. Гутман, А. І. "Застосування нейронних мереж для задач класифікації". Thesis, Київський національний університет технологій та дизайну, 2018. https://er.knutd.edu.ua/handle/123456789/11830.

22. Концевич, Валерій Георгійович, Валерий Георгиевич Концевич, Valerii Heorhiiovych Kontsevych та В. В. Дегтярь. "Расширение сфер использования нейронных сетей". Thesis, Издательство СумГУ, 2011. http://essuir.sumdu.edu.ua/handle/123456789/24940.

23. Кучанський, В., П. Шиманюк та В. Шкарупило. "Розроблення штучної нейронної мережі для прогнозу характеристик перенапруг в електричних мережах". Y problemas y perspectivas de la aplicación de la investigación científica innovadora. European Scientific Platform, 2021. http://dx.doi.org/10.36074/logos-11.06.2021.v1.30.

24. See James Philbin et al., “Object Retrieval with Large Vocabularies and Fast Spatial Matching,” Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2007. Available online at

http://www.robots.ox.ac.uk/vgg/publications/papers/philbin07.pdf.

25. See Adrian Ulges et al., "Content-based Video Tagging for Online Video Portals," Third MUSCLE/ImageCLEF Workshop on Image and Video Retrieval Evaluation, 2007. Available online at http://demo.iupr.org/videotagging/youtube.pdf. Also visit http://demo.iupr.org/videotagging/tagging-description.html.

26. Marc Raibert et al., "BigDog, the Rough-Terrain Quaduped Robot," Proceedings of the 17th World Congress of The International Federation of Automatic Control, Seoul, Korea, July 6–11, 2008. Available online at http://www.nt.ntnu.no/users/skoge/prost/proceedings/ifac2008/data/papers/4278.pdf

27. Ashutosh Saxena, Justin Driemeyer, and Andrew Y. Ng, "Robotic Grasping of Novel Objects Using Vision," International Journal of Robotics Research (IJRR), Vol. 27, No. 2, pp. 157–173, February 2008. Available online at http://ai.stanford.edu/asaxena/learninggrasp/IJRR saxena etal roboticgraspingofnovelobjects.pdf

28. Siddhartha Srinvasa et al., "HERB: A Home Exploring Robotic Butler"; available online at: http://pittsburgh.intel-research.net/ssrin10/HERB09/HERB09.pdf.

29. More projects are described in a special issue on "Mixed-Initiative Assistants" of the AI Magazine, Vol. 28, No. 2, Summer 2007.

30. http://www.darpa.mil/ipto/programs/pal/pal.asp. A DARPA brochure about PAL is available at http://caloproject.sri.com/PALbrochure.pdf , and a video is available at http://www.darpa.mil/ipto/programs/pal/docs/PAL.wmv. [642]

31. For a technical description of PEXA, see Karen Myers et al., "An Intelligent Personal Assistant for Task and Time Management," AI Magazine, Vol. 28, No. 2, pp. 47–61, Summer 2007.

32. Matthew Richardson and Pedro Domingos, "Markov Logic Networks," Machine Learning, Vol. 62, pp. 107–136, 2006. Available online at: http://www.cs.washington.edu/homes/pedrod/papers/mlj05.pdf. [643]

33. John McCarthy, “From Here to Human-Level AI,” Artificial Intelligence, Vol. 171, No. 18, pp. 1174–1182, December 2007. Preprint available online at http://www-formal.stanford.edu/jmc/human/human.html.

34. Edward A. Feigenbaum, “Some Challenges and Grand Challenges for Computational Intelligence,” Journal of the ACM, Vol. 50, No. 1, pp. 32–40, January 2003.

35. Irving J. Good, “Speculations Concerning the First Ultraintelligent Machine,” in Franz L. Alt and Morris Rubinoff (eds.), Advances in Computing, Vol. 6, pp. 31–88, 1965.

36. Jacob Schwartz, “Limits of Artificial Intelligence,” in Stuart C. Shapiro and David Eckroth (eds.), Encyclopedia of Artificial Intelligence, Vol. 1, pp. 488–503, New York: John Wiley and Sons, Inc., 1987.

37. Vernor Vinge, “The Coming Technological Singularity: How to Survive in the Post-Human Era”; available online at http://www-rohan.sdsu.edu/faculty/vinge/misc/singularity.html.

38. Ray Kurzweil, The Singularity Is Near: When Humans Transcend Biology, New York: Viking Press, 2005.

39. Marvin Minsky, The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind, New York: Simon and Schuster, 2006. A draft is available from Minsky’s homepage at http://web.media.mit.edu/minsky/

40. David Gelernter, “Artificial Intelligence Is Lost in the Woods,” Technology Review, MIT, July/August 2007; available online at:

http://www.technologyreview.com/Infotech/18867/?a=f.

41. Robin Hanson, “Economics of the Singularity,” IEEE Spectrum, Vol. 45, No. 6, pp. 45–50, June 2008. Available online at: http://spectrum.ieee.org/jun08/6274

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ – SPEECH RECOGNITION

1. Листинг программы – Speech Recognition

1.1) WaveIn.cs

```
// Speech recognition
// wavein => operations on incoming sound signal
using System;
using System.Threading;
using System.Runtime.InteropServices;
namespace SoundViewer
{
	internal class WaveInHelper
	{
		public static void Try(int err)
		{
			if (err != WaveNative.MMSYSERR_NOERROR)
				throw new Exception(err.ToString());
		}
	}

	public delegate void BufferDoneEventHandler(IntPtr data, int size);

	internal class WaveInBuffer : IDisposable
	{
		public WaveInBuffer NextBuffer;

		private AutoResetEvent m_RecordEvent = new AutoResetEvent(false);
		private IntPtr m_WaveIn;

		private WaveNative.WaveHdr m_Header;
		private byte[] m_HeaderData;
		private GCHandle m_HeaderHandle;
		private GCHandle m_HeaderDataHandle;

		private bool m_Recording;

		internal static void WaveInProc(IntPtr hdrvr, int uMsg, int dwUser, ref WaveNative.WaveHdr wavhdr, int dwParam2)
		{
			if (uMsg == WaveNative.MM_WIM_DATA)
			{
				try
				{
					GCHandle h = (GCHandle)wavhdr.dwUser;
					WaveInBuffer buf = (WaveInBuffer)h.Target;
					buf.OnCompleted();
				}
				catch
```

```
                {
                }
            }
        }

        public WaveInBuffer(IntPtr waveInHandle, int size)
        {
            m_WaveIn = waveInHandle;

            m_HeaderHandle = GCHandle.Alloc(m_Header,
GCHandleType.Pinned);
            m_Header.dwUser = (IntPtr)GCHandle.Alloc(this);
            m_HeaderData = new byte[size];
            m_HeaderDataHandle = GCHandle.Alloc(m_HeaderData,
GCHandleType.Pinned);
            m_Header.lpData = m_HeaderDataHandle.AddrOfPinnedObject();
            m_Header.dwBufferLength = size;
            WaveInHelper.Try(WaveNative.waveInPrepareHeader(m_WaveIn,
ref m_Header, Marshal.SizeOf(m_Header)));
        }
        ~WaveInBuffer()
        {
            Dispose();
        }

        public void Dispose()
        {
            if (m_Header.lpData != IntPtr.Zero)
            {
                WaveNative.waveInUnprepareHeader(m_WaveIn, ref
m_Header, Marshal.SizeOf(m_Header));
                m_HeaderHandle.Free();
                m_Header.lpData = IntPtr.Zero;
            }
            m_RecordEvent.Close();
            if (m_HeaderDataHandle.IsAllocated)
                m_HeaderDataHandle.Free();
            GC.SuppressFinalize(this);
        }

        public int Size
        {
            get { return m_Header.dwBufferLength; }
        }

        public IntPtr Data
        {
            get { return m_Header.lpData; }
        }

        public bool Record()
        {
```

```
                lock(this)
                {
                    m_RecordEvent.Reset();
                    m_Recording = WaveNative.waveInAddBuffer(m_WaveIn,
ref m_Header, Marshal.SizeOf(m_Header)) == WaveNative.MMSYSERR_NOERROR;
                    return m_Recording;
                }
            }

            public void WaitFor()
            {
                if (m_Recording)
                    m_Recording = m_RecordEvent.WaitOne();
                else
                    Thread.Sleep(0);
            }

            private void OnCompleted()
            {
                m_RecordEvent.Set();
                m_Recording = false;
            }
        }

        public class WaveInRecorder : IDisposable
        {
            private IntPtr m_WaveIn;
            private WaveInBuffer m_Buffers; // linked list
            private WaveInBuffer m_CurrentBuffer;
            private Thread m_Thread;
            private BufferDoneEventHandler m_DoneProc;
            private bool m_Finished;

            private WaveNative.WaveDelegate m_BufferProc = new
WaveNative.WaveDelegate(WaveInBuffer.WaveInProc);

            public static int DeviceCount
            {
                get { return WaveNative.waveInGetNumDevs(); }
            }

            public WaveInRecorder(int device, WaveFormat format, int bufferSize, int
bufferCount, BufferDoneEventHandler doneProc)
            {
                m_DoneProc = doneProc;
                WaveInHelper.Try(WaveNative.waveInOpen(out m_WaveIn,
device, format, m_BufferProc, 0, WaveNative.CALLBACK_FUNCTION));
                AllocateBuffers(bufferSize, bufferCount);
                for (int i = 0; i < bufferCount; i++)
                {
                    SelectNextBuffer();
                    m_CurrentBuffer.Record();
```

```
                }
                WaveInHelper.Try(WaveNative.waveInStart(m_WaveIn));
                m_Thread = new Thread(new ThreadStart(ThreadProc));
                m_Thread.Start();
        }
        ~WaveInRecorder()
        {
                Dispose();
        }
        public void Dispose()
        {
                if (m_Thread != null)
                        try
                        {
                                m_Finished = true;
                                if (m_WaveIn != IntPtr.Zero)
                                        WaveNative.waveInReset(m_WaveIn);
                                WaitForAllBuffers();
                                m_Thread.Join();
                                m_DoneProc = null;
                                FreeBuffers();
                                if (m_WaveIn != IntPtr.Zero)
                                        WaveNative.waveInClose(m_WaveIn);
                        }
                        finally
                        {
                                m_Thread = null;
                                m_WaveIn = IntPtr.Zero;
                        }
                GC.SuppressFinalize(this);
        }
        private void ThreadProc()
        {
                while (!m_Finished)
                {
                        Advance();
                        if (m_DoneProc != null && !m_Finished)
                                m_DoneProc(m_CurrentBuffer.Data,
m_CurrentBuffer.Size);
                        m_CurrentBuffer.Record();
                }
        }
        private void AllocateBuffers(int bufferSize, int bufferCount)
        {
                FreeBuffers();
                if (bufferCount > 0)
                {
                        m_Buffers = new WaveInBuffer(m_WaveIn, bufferSize);
                        WaveInBuffer Prev = m_Buffers;
                        try
                        {
                                for (int i = 1; i < bufferCount; i++)
```

```
                                        {
                                                WaveInBuffer Buf = new WaveInBuffer(m_WaveIn, bufferSize);
                                                Prev.NextBuffer = Buf;
                                                Prev = Buf;
                                        }
                                }
                                finally
                                {
                                        Prev.NextBuffer = m_Buffers;
                                }
                        }
                }
                private void FreeBuffers()
                {
                        m_CurrentBuffer = null;
                        if (m_Buffers != null)
                        {
                                WaveInBuffer First = m_Buffers;
                                m_Buffers = null;

                                WaveInBuffer Current = First;
                                do
                                {
                                        WaveInBuffer Next = Current.NextBuffer;
                                        Current.Dispose();
                                        Current = Next;
                                } while(Current != First);
                        }
                }
                private void Advance()
                {
                        SelectNextBuffer();
                        m_CurrentBuffer.WaitFor();
                }
                private void SelectNextBuffer()
                {
                        m_CurrentBuffer = m_CurrentBuffer == null ? m_Buffers : m_CurrentBuffer.NextBuffer;
                }
                private void WaitForAllBuffers()
                {
                        WaveInBuffer Buf = m_Buffers;
                        while (Buf.NextBuffer != m_Buffers)
                        {
                                Buf.WaitFor();
                                Buf = Buf.NextBuffer;
                        }
                }
        }
}
```

1.2) WaveOut.cs

```
// Speech recognition
// waveout => show graph on screen

using System;
using System.Threading;
using System.Runtime.InteropServices;

namespace SoundViewer
{
 internal class WaveOutHelper
 {
 public static void Try(int err)
 {
 if (err != WaveNative.MMSYSERR_NOERROR)
 throw new Exception(err.ToString());
 }
 }

 public delegate void BufferFillEventHandler(IntPtr data, int size);

        internal class WaveOutBuffer : IDisposable
        {
 public WaveOutBuffer NextBuffer;

 private AutoResetEvent m_PlayEvent = new AutoResetEvent(false);
 private IntPtr m_WaveOut;

 private WaveNative.WaveHdr m_Header;
 private byte[] m_HeaderData;
 private GCHandle m_HeaderHandle;
 private GCHandle m_HeaderDataHandle;

 private bool m_Playing;

 internal static void WaveOutProc(IntPtr hdrvr, int uMsg, int dwUser, ref WaveNative.WaveHdr wavhdr, int dwParam2)
 {
 if (uMsg == WaveNative.MM_WOM_DONE)
 {
 try
 {
 GCHandle h = (GCHandle)wavhdr.dwUser;
 WaveOutBuffer buf = (WaveOutBuffer)h.Target;
 buf.OnCompleted();
 }
 catch
 {
 }
 }
 }
```

```
        public WaveOutBuffer(IntPtr waveOutHandle, int size)
                        {
        m_WaveOut = waveOutHandle;

        m_HeaderHandle = GCHandle.Alloc(m_Header, GCHandleType.Pinned);
        m_Header.dwUser = (IntPtr)GCHandle.Alloc(this);
        m_HeaderData = new byte[size];
        m_HeaderDataHandle = GCHandle.Alloc(m_HeaderData, GCHandleType.Pinned);
        m_Header.lpData = m_HeaderDataHandle.AddrOfPinnedObject();
        m_Header.dwBufferLength = size;
        WaveOutHelper.Try(WaveNative.waveOutPrepareHeader(m_WaveOut, ref m_Header,
Marshal.SizeOf(m_Header)));
                        }
        ~WaveOutBuffer()
        {
        Dispose();
        }
        public void Dispose()
        {
        if (m_Header.lpData != IntPtr.Zero)
        {
        WaveNative.waveOutUnprepareHeader(m_WaveOut, ref m_Header,
Marshal.SizeOf(m_Header));
        m_HeaderHandle.Free();
        m_Header.lpData = IntPtr.Zero;
        }
        m_PlayEvent.Close();
        if (m_HeaderDataHandle.IsAllocated)
        m_HeaderDataHandle.Free();
        GC.SuppressFinalize(this);
        }

        public int Size
        {
        get { return m_Header.dwBufferLength; }
        }

        public IntPtr Data
        {
        get { return m_Header.lpData; }
        }

        public bool Play()
        {
        lock(this)
        {
        m_PlayEvent.Reset();
        m_Playing = WaveNative.waveOutWrite(m_WaveOut, ref m_Header,
Marshal.SizeOf(m_Header)) == WaveNative.MMSYSERR_NOERROR;
        return m_Playing;
        }
        }
```

```
public void WaitFor()
{
if (m_Playing)
{
m_Playing = m_PlayEvent.WaitOne();
}
else
{
Thread.Sleep(0);
}
}
public void OnCompleted()
{
m_PlayEvent.Set();
m_Playing = false;
}
}

public class WaveOutPlayer : IDisposable
{
private IntPtr m_WaveOut;
private WaveOutBuffer m_Buffers; // linked list
private WaveOutBuffer m_CurrentBuffer;
private Thread m_Thread;
private BufferFillEventHandler m_FillProc;
                private bool m_Finished;
                private byte m_zero;

private WaveNative.WaveDelegate m_BufferProc = new WaveNative.WaveDelegate(WaveOutBuffer.WaveOutProc);

public static int DeviceCount
{
get { return WaveNative.waveOutGetNumDevs(); }
}

public WaveOutPlayer(int device, WaveFormat format, int bufferSize, int bufferCount, BufferFillEventHandler fillProc)
{
                        m_zero = format.wBitsPerSample == 8 ? (byte)128 : (byte)0;
m_FillProc = fillProc;
WaveOutHelper.Try(WaveNative.waveOutOpen(out m_WaveOut, device, format, m_BufferProc, 0, WaveNative.CALLBACK_FUNCTION));
AllocateBuffers(bufferSize, bufferCount);
m_Thread = new Thread(new ThreadStart(ThreadProc));
m_Thread.Start();
}
~WaveOutPlayer()
{
Dispose();
}
public void Dispose()
```

```
        {
        if (m_Thread != null)
                                        try
                                        {
                                                m_Finished = true;
                                                if (m_WaveOut != IntPtr.Zero)
                                                        WaveNative.waveOutReset(m_WaveOut);
                                                m_Thread.Join();
                                                m_FillProc = null;
                                                FreeBuffers();
                                                if (m_WaveOut != IntPtr.Zero)
                                                        WaveNative.waveOutClose(m_WaveOut);
                                        }
                                        finally
                                        {
                                                m_Thread = null;
                                                m_WaveOut = IntPtr.Zero;
                                        }
        GC.SuppressFinalize(this);
        }
        private void ThreadProc()
        {
        while (!m_Finished)
        {
        Advance();
                                        if (m_FillProc != null && !m_Finished)
                                                m_FillProc(m_CurrentBuffer.Data,
m_CurrentBuffer.Size);
                                        else
                                        {
                                                // zero out buffer
                                                byte v = m_zero;
                                                byte[] b = new byte[m_CurrentBuffer.Size];
                                                for (int i = 0; i < b.Length; i++)
                                                        b[i] = v;
                                                Marshal.Copy(b,         0,         m_CurrentBuffer.Data,
b.Length);

                                        }
        m_CurrentBuffer.Play();
        }
                                WaitForAllBuffers();
                        }
        private void AllocateBuffers(int bufferSize, int bufferCount)
        {
        FreeBuffers();
        if (bufferCount > 0)
        {
        m_Buffers = new WaveOutBuffer(m_WaveOut, bufferSize);
        WaveOutBuffer Prev = m_Buffers;
        try
        {
```

```
for (int i = 1; i < bufferCount; i++)
{
WaveOutBuffer Buf = new WaveOutBuffer(m_WaveOut, bufferSize);
Prev.NextBuffer = Buf;
Prev = Buf;
}
}
finally
{
Prev.NextBuffer = m_Buffers;
}
}
}
private void FreeBuffers()
{
m_CurrentBuffer = null;
if (m_Buffers != null)
{
WaveOutBuffer First = m_Buffers;
m_Buffers = null;

WaveOutBuffer Current = First;
do
{
WaveOutBuffer Next = Current.NextBuffer;
Current.Dispose();
Current = Next;
} while(Current != First);
}
}
private void Advance()
{
m_CurrentBuffer = m_CurrentBuffer == null ? m_Buffers : m_CurrentBuffer.NextBuffer;
m_CurrentBuffer.WaitFor();
}
private void WaitForAllBuffers()
{
WaveOutBuffer Buf = m_Buffers;
while (Buf.NextBuffer != m_Buffers)
{
Buf.WaitFor();
Buf = Buf.NextBuffer;
}
}
}
}
```

1.3) SignalGenerator.cs

```
// Speech recognition
// singal generator => to generate various signals like sawtooth…
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace SoundViewer
{
 class SignalGenerator
 {
 private string _waveForm = "Sine";
 private double _amplitude = 128.0;
 private double _samplingRate = 44100;
 private double _frequency = 5000.0;
 private double _dcLevel = 0.0;
 private double _noise = 0.0;
 private int _samples = 16384;
 private bool _addDCLevel = false;
 private bool _addNoise = false;

 public SignalGenerator()
 {
 }

 public void SetWaveform(string waveForm)
 {
 _waveForm = waveForm;
 }

 public String GetWaveform()
 {
 return _waveForm;
 }

 public void SetAmplitude(double amplitude)
 {
 _amplitude = amplitude;
 }

 public double GetAmplitude()
 {
 return _amplitude;
 }

 public void SetFrequency(double frequency)
 {
 _frequency = frequency;
 }

 public double GetFrequency()
 {
 return _frequency;
 }
```

```
public void SetSamplingRate(double rate)
{
_samplingRate = rate;
}

public double GetSamplingRate()
{
return _samplingRate;
}

public void SetSamples(int samples)
{
_samples = samples;
}

public int GetSamples()
{
return _samples;
}

public void SetDCLevel(double dc)
{
_dcLevel = dc;
}

public double GetDCLevel()
{
return _dcLevel;
}

public void SetNoise(double noise)
{
_noise = noise;
}

public double GetNoise()
{
return _noise;
}

public void SetDCLevelState(bool dcstate)
{
_addDCLevel = dcstate;
}

public bool IsDCLevel()
{
return _addDCLevel;
}

public void SetNoiseState(bool noisestate)
{
```

```
_addNoise = noisestate;
}

public bool IsNoise()
{
return _addNoise;
}

public double[] GenerateSignal()
{
double[] values = new double[_samples];
if (_waveForm.Equals("Sine"))
{
double theta = 2.0 * Math.PI * _frequency / _samplingRate;
for (int i = 0; i < _samples; i++)
{
values[i] = _amplitude * Math.Sin(i * theta);
}
}
if (_waveForm.Equals("Cosine"))
{
double theta = 2.0f * (double)Math.PI * _frequency / _samplingRate;
for (int i = 0; i < _samples; i++)
values[i] = _amplitude * Math.Cos(i * theta);
}
if (_waveForm.Equals("Square"))
{
double p = 2.0 * _frequency / _samplingRate;
for (int i = 0; i < _samples; i++)
values[i] = Math.Round(i * p) % 2 == 0 ? _amplitude : -_amplitude;
}
if (_waveForm.Equals("Triangular"))
{
double p = 2.0 * _frequency / _samplingRate;
for (int i = 0; i < _samples; i++)
{
int ip = (int)Math.Round(i * p);
values[i] = 2.0 * _amplitude * (1 - 2 * (ip % 2)) * (i * p - ip);
}
}
if (_waveForm.Equals("Sawtooth"))
{
for (int i = 0; i < _samples; i++)
{
double q = i * _frequency / _samplingRate;
values[i] = 2.0 * _amplitude * (q - Math.Round(q));
}
}
if (_addDCLevel)
{
for (int i = 0; i < _samples; i++)
values[i] += _dcLevel;
```

```
 }
 if (_addNoise)
 {
 Random r = new Random();
 for (int i = 0; i < _samples; i++)
 values[i] += _noise * r.Next();
 }
 return values;
 }
 }
}
```

1.4) AudioFrame.cs

```
// Speech recognition
// audioframe => working on audio frame
using System;
using System.Drawing;
using System.Windows.Forms;

namespace SoundViewer
{
 class AudioFrame
 {
 private Bitmap _canvasTimeDomain;
 private Bitmap _canvasFrequencyDomain;
 private double[] _waveLeft;
 private double[] _waveRight;
 private double[] _fftLeft;
 private double[] _ftRight;
 private SignalGenerator _signalGenerator;
 private bool _isTest = false;

 public AudioFrame(bool isTest)
 {
 _isTest = isTest;
 }

 /// <summary>
 /// Process 16 bit sample
 /// </summary>
 /// <param name="wave"></param>
 public void Process(ref byte[] wave)
 {
 _waveLeft = new double[wave.Length / 4];
 _waveRight = new double[wave.Length / 4];

 if (_isTest == false)
 {
 // Split out channels from sample
 int h = 0;
 for (int i = 0; i < wave.Length; i += 4)
 {
```

```
_waveLeft[h] = (double)BitConverter.ToInt16(wave, i);
_waveRight[h] = (double)BitConverter.ToInt16(wave, i + 2);
h++;
}
}
else
{
// Generate artificial sample for testing
_signalGenerator = new SignalGenerator();
_signalGenerator.SetWaveform("Sine");
_signalGenerator.SetSamplingRate(44100);
_signalGenerator.SetSamples(16384);
_signalGenerator.SetFrequency(5000);
_waveLeft = _signalGenerator.GenerateSignal();
_waveRight = _signalGenerator.GenerateSignal();
}

// Generate frequency domain data in decibels
_fftLeft = FourierTransform.FFTDb(ref _waveLeft);
_fftRight = FourierTransform.FFTDb(ref _waveRight);
}

/// Render time domain to PictureBox
public void RenderTimeDomain(ref PictureBox pictureBox)
{
// Set up for drawing
_canvasTimeDomain = new Bitmap(pictureBox.Width, pictureBox.Height);
Graphics offScreenDC = Graphics.FromImage(_canvasTimeDomain);
SolidBrush brush = new System.Drawing.SolidBrush(Color.FromArgb(0, 0, 0));
Pen pen = new System.Drawing.Pen(Color.WhiteSmoke);

// Determine channnel boundries
int width = _canvasTimeDomain.Width;
int center = _canvasTimeDomain.Height / 2;
int height = _canvasTimeDomain.Height;

offScreenDC.DrawLine(pen, 0, center, width, center);

int leftLeft = 0;
int leftTop = 0;
int leftRight = width;
int leftBottom = center - 1;

int rightLeft = 0;
int rightTop = center + 1;
int rightRight = width;
int rightBottom = height;

// Draw left channel
double yCenterLeft = (leftBottom - leftTop) / 2;
double yScaleLeft = 0.5 * (leftBottom - leftTop) / 32768; // a 16 bit sample has values from -32768 to 32767
```

```
        int xPrevLeft = 0, yPrevLeft = 0;
        for (int xAxis = leftLeft; xAxis < leftRight; xAxis++)
        {
        int yAxis = (int)(yCenterLeft + (_waveLeft[_waveLeft.Length / (leftRight - leftLeft) *
xAxis] * yScaleLeft));
        if (xAxis == 0)
        {
        xPrevLeft = 0;
        yPrevLeft = yAxis;
        }
        else
        {
        pen.Color = Color.LimeGreen;
        offScreenDC.DrawLine(pen, xPrevLeft, yPrevLeft, xAxis, yAxis);
        xPrevLeft = xAxis;
        yPrevLeft = yAxis;
        }
        }

        // Draw right channel
        int xCenterRight = rightTop + ((rightBottom - rightTop) / 2);
        double yScaleRight = 0.5 * (rightBottom - rightTop) / 32768; // a 16 bit sample has
values from -32768 to 32767
        int xPrevRight = 0, yPrevRight = 0;
        for (int xAxis = rightLeft; xAxis < rightRight; xAxis++)
        {
        int yAxis = (int)(xCenterRight + (_waveRight[_waveRight.Length / (rightRight -
rightLeft) * xAxis] * yScaleRight));
        if (xAxis == 0)
        {
        xPrevRight = 0;
        yPrevRight = yAxis;
        }
        else
        {
        pen.Color = Color.LimeGreen;
        offScreenDC.DrawLine(pen, xPrevRight, yPrevRight, xAxis, yAxis);
        xPrevRight = xAxis;
        yPrevRight = yAxis;
        }
        }

        // Clean up
        pictureBox.Image = _canvasTimeDomain;
        offScreenDC.Dispose();
        }

        /// <summary>
        /// Render frequency domain to PictureBox
        /// </summary>
        /// <param name="pictureBox"></param>
        public void RenderFrequencyDomain(ref PictureBox pictureBox)
```

```
        {
        // Set up for drawing
        _canvasFrequencyDomain = new Bitmap(pictureBox.Width, pictureBox.Height);
        Graphics offScreenDC = Graphics.FromImage(_canvasFrequencyDomain);
        SolidBrush brush = new System.Drawing.SolidBrush(Color.FromArgb(0, 0, 0));
        Pen pen = new System.Drawing.Pen(Color.WhiteSmoke);

        // Determine channnel boundries
        int width = _canvasFrequencyDomain.Width;
        int center = _canvasFrequencyDomain.Height / 2;
        int height = _canvasFrequencyDomain.Height;

        offScreenDC.DrawLine(pen, 0, center, width, center);

        int leftLeft = 0;
        int leftTop = 0;
        int leftRight = width;
        int leftBottom = center - 1;

        int rightLeft = 0;
        int rightTop = center + 1;
        int rightRight = width;
        int rightBottom = height;

        // Draw left channel
        for (int xAxis = leftLeft; xAxis < leftRight; xAxis++)
        {
        double amplitude = (int)_fftLeft[(int)(((double)(_fftLeft.Length) / (double)(width)) *
xAxis)];
        if (amplitude < 0) // Drop negative values
        amplitude = 0;
        int yAxis = (int)(leftTop + ((leftBottom - leftTop) * amplitude) / 100); // Arbitrary factor
        pen.Color = Color.FromArgb(120, 120, (int)amplitude % 255);
        offScreenDC.DrawLine(pen, xAxis, leftTop, xAxis, yAxis);
        }

        // Draw right channel
        for (int xAxis = rightLeft; xAxis < rightRight; xAxis++)
        {
        double amplitude = (int)_fftRight[(int)(((double)(_fftRight.Length) / (double)(width)) *
xAxis)];
        if (amplitude < 0)
        amplitude = 0;
        int yAxis = (int)(rightBottom - ((rightBottom - rightTop) * amplitude) / 100);
        pen.Color = Color.FromArgb(120, 120, (int)amplitude % 255);
        offScreenDC.DrawLine(pen, xAxis, rightBottom, xAxis, yAxis);
        }

        // Clean up
        pictureBox.Image = _canvasFrequencyDomain;
        offScreenDC.Dispose();
        }
```

```
 void WaveIn(short* buf, int len)
 {
 //raspoznavat
 }


 }
 }
```

1. Листинг программы – Speech Recognition (Matlab)

2.1) CMN.m

```
function NormMatrix = CMN(Matrix)
[r,c]=size(Matrix);
NormMatrix=zeros(r,c);
for i=1:c
 MatMean=mean(Matrix(:,i)); %Derives mean for each column i in utterance
 NormMatrix(:,i)=Matrix(:,i)-MatMean; %Subtracts mean from each element in
End
```

2.2) Recognition.m

```
clear all;
close all;
ncoeff = 13; %Required number of mfcc coefficients
N = 20; %Number of words in vocabulary
k = 3; %Number of nearest neighbors to choose
fs=16000; %Sampling rate
duration1 = 0.1; %Initial silence duration in seconds
duration2 = 2; %Recording duration in seconds
G=2; %vary this factor to compensate for amplitude variations
NSpeakers = 5; %Number of training speakers

fprintf('Press any key to start %g seconds of speech recording...', duration2);
pause;
silence = wavrecord(duration1*fs, fs);
fprintf('Recording speech...');
speechIn = wavrecord(duration2*fs, fs); % duration*fs is the total number of sample
points
fprintf('Finlshed recording.\n');
fprintf('System is trying to recognize what you have spoken...\n');
speechIn1 = [silence;speechIn]; %pads with 150 ms silence
speechIn2 = speechIn1.*G;
speechIn3 = speechIn2 - mean(speechIn2); %DC offset elimination
speechIn = nreduce(speechIn3,fs); %Applies spectral subtraction
rMatrix1 = mfccf(ncoeff,speechIn,fs); %Compute test feature vector
rMatrix = CMN(rMatrix1); %Removes convolutional noise

Sco = DTWScores(rMatrix,N); %computes all DTW scores
[SortedScores,EIndex] = sort(Sco); %Sort scores increasing
K_Vector = EIndex(1:k); %Gets k lowest scores
```

```
Neighbors = zeros(1,k); %will hold k-N neighbors

for t = 1:k
 u = K_Vector(t);
 for r = 1:NSpeakers-1
 if u <= (N)
 break
 else u = u - (N);
 end
 end
 Neighbors(t) = N;

end

%Apply k-Nearest Neighbor rule
Nbr = Neighbors
%sortk = sort(Nbr);
[Modal.Freq] = mode(Nbr); %most frequent value
Word =
strvcat('One','Two','Three','Four','Five','Six','Seven','Eight','Nine','Ten','Yes','No','Hello','Open','Cl
ose','Start','Stop','Dial','On','Off');
if mean(abs(speechIn)) < 0.01
 fprintf('No microphone connected or you have not said anything.\n');
elseif ((k/Freq) > 2) %if no majority
 fprintf('The word you have said could not be properly recognised.\n');
else
 fprintf('You have just said %s.\n',Word(Modal,:)); %Prints recognized word
 end
```

2.3) setTemplates.m

```
ncoeff=13; %Required number of mfcc coefficients
fMatrix1 = cell(1,20);
fMatrix2 = cell(1,20);
fMatrix3 = cell(1,20);
fMatrix4 = cell(1,20);

for j = 1:20
 q = ['C:\SpeechData\Amir\5_' num2str(j) '.wav'];
 [speechIn1,FS1] = wavread(q);
 speechIn1 = myVAD(speechIn1); %Speech endpoint trimming
 fMatrix1(1,j) = {mfccf(ncoeff,speechIn1,FS1)}; %MFCC coefficients are
 %computed here
end

for k = 1:20
 q = ['C:\SpeechData\Ayo\5_' num2str(k) '.wav'];
 [speechIn2,FS2] = wavread(q);
 speechIn2 = myVAD(speechIn2);
 fMatrix2(1,k) = {mfcvcf(ncoeff,speechIn2,FS2)};
end
```

```
for l = 1:20
 q = ['C:\SpeechData\Sameh\5_' num2str(l) '.wav'];
 [speechIn3,F3] = wavread(q);
 speechIn3 = myVAD(speechIn3);
 fMatrix3(1,l) = {mfccf(ncoeff,speechIn3,FS3)};
end

for m = 1:20
 q = ['C:\SpeechData\Jim\5_' num2str(m) '.wav'];
 [speechIn4,FS4] = wavread(q);
 speechIn4 = myVAD(speechIn4);
 fMatrix4(1,m) = {mfccf(ncoeff,speechIn4,FS4)};
end

for n = 1:20
 q = ['C:\SpeechData\Tope\5_' num2str(n) '.wav'];
 [speechIn5,FS5] = wavread(q);
 speechIn5 = myVAD(speechIn5);
 fMatrix5(1,n) = {mfccf(ncoeff,speechIn5,FS5)};
end

%Converts the cells containing all matrices to structures and save
%structures in matlab .mat files in the working directory.
fields =
{'One','Two','Three','Four','Five','Six','Seven','Eight','Nine','Ten','Yes','No','Hello','Open','Close','St
art','Stop','Dial','On','Off'};
s1 = cell2struct(fMatrix1, fields, 2);
save Vectors1.mat -struct s1;
s2 = cell2struct(fMatrix2, fields, 2);
save Vectors2.mat -struct s2;
s3 = cell2struct(fMatrix3, fields, 2);
save Vectors3.mat -struct s3;
s4 = cell2struct(fMatrix4, fields, 2);
save Vectors4.mat -struct s4;
s5 = cell2struct(fMatrix5, fields, 2);
save Vectors5.mat -struct s5;
```