Міністерство освіти і науки України

Харківський національний університет імені В.Н. Каразіна

Факультет комп'ютерних наук

Спеціальність 125 «Кібербезпека»

Освітня програма «Безпека інформаційних та комунікаційних систем»

«Допущено до захисту»

Зав. кафедрою БІСТ

Рассомахін С.Г.

_____

« » 2022р.

Пояснювальна записка

до кваліфікаційної роботи магістра

на тему: «Дослідження методів глибинного навчання та їх застосування в лінгвістичній стеганографії»

оцінка « » Керівник проф. Кузнецов О. О.

Голова ЕК Рецензент проф. Толстолузька О. Г.

Доценко С.І. _____ Виконавець : студентка групи КБ-61

Попенко В. О.

Харків – 2022

РЕФЕРАТ

Кваліфікаційна робота магістра, 70 с., 16 рис., 7 табл., 2 формули, 32 джерела.

Робота містить вступ, 4 розділи, висновки, список використаної літератури та 3 додатки.

У роботі проведено дослідження технології глибинного навчання. Розглянуто позицію технології глибинного навчання в сфері штучного інтелекту, проведено аналіз її властивостей та залежностей, досліджено методи глибинного навчання та їх застосунки. Проведено дослідження методів глибинного навчання для обробки природної мови та їх використання в методах лінгвістичної стеганографії.

Мета роботи полягала в аналізі властивостей та залежностей технології глибинного навчання, дослідженні методів та застосунків глибинного навчання та аналізі існуючих реалізацій обробки природної мови, з використанням глибинного навчання, а також власна реалізація мовою програмування Python лінгвістичної стеганосистеми з використанням глибинного навчання, а саме рекурентної нейронної мережі, яка використовується для автоматичної генерації текстових контейнерів високої якості для приховування таємного потоку бітів, та її експериментальні дослідження щодо таких характеристик як швидкість вбудовування, ефективність вбудовування, непомітність вбудовування, ємність для вбудовування. В процесі генерації було запропоновано кодування фіксованої довжини та кодування змінної величини для кодування слів на основі їхнього умовного розподілу ймовірності. Розроблена програмна реалізація дозволяє використовувати методи лінгвістичної стеганографії більш ефективно.

Ключові слова: ПРИХОВУВАННЯ ІНФОРМАЦІЇ, ЛІНГВІСТИЧНА СТЕГАНОГРАФІЯ, СТЕГАНОГРАФІЯ, ШТУЧНИЙ ІНТЕЛЕКТ, МАШИННЕ НАВЧАННЯ, ГЛИБИННЕ НАВЧАННЯ, РЕКУРЕНТНА НЕЙРОННА МЕРЕЖА, ОБРОБКА ПРИРОДНОЇ МОВИ, АВТОМАТИЧНА ГЕНЕРАЦІЯ ТЕКСТУ, PYTHON

ABSTRACT

Qualifying work of master: 70 pages, 16 images, 7 tables, 2 equations, 32 references.

The work consists of the introduction, 4 sections, conclusions, a list of references and 3 applications.

The work conducted a study of deep learning technology. The position of deep learning technology in the field of artificial intelligence was considered, its properties and dependencies were analyzed, deep learning methods and their application were investigated. A study of deep learning methods for natural language processing and their usage in linguistic steganography methods was conducted.

The purpose of the work was to analyze the properties and dependencies of deep learning technology, to research methods and applications of deep learning and to analyze existing implementations of natural language processing using deep learning, as well as own implementation in the Python programming language of a linguistic steganosystem using deep learning, specifically, a recurrent neural network, which is used to automatically generate high-quality text containers to hide a secret bit stream, and its experimental studies on characteristics such as embedding rate, information embedding efficiency, information imperceptibility and information hidden capacity. In the generation process, fixed-length encoding and variable-value encoding were proposed to encode words based on their conditional probability distribution. The developed software implementation allows you to use linguistic steganography methods more effectively.

Key words: INFORMATION HIDING, LINGUISTIC STEGANOGRAPHY, STEGANOGRAPHY, ARTIFICIAL INTELLIGENCE, MACHINE LEARNING, DEEP LEARNING, RECURRENT NEURAL NETWORK, NATURAL LANGUAGE PROCESSING, AUTOMATIC TEXT GENERATION, PYTHON

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАЧЕНЬ

ГН – Глибинне навчання
ШІ – Штучний інтелект
МН – Машинне навчання
MLP – Multi-Layer Perceptron
CNN – Convolutional Neural Network
DNN – Deep Neural Network
RNN – Recurrent Neural Network
ANN – Artificial Neural Network
ReLU – Rectified Linear Unit
SGD – Stochastic Gradient Descent
VGG – Visual Geometry Group
GRU – Gated Recurrent Unit
GAN – Generative Adversarial Network
AE – Autoencoder
RBM – Restricted Boltzmann Machine
SOM – Self-Organizing Map
DBN – Deep Belief Network
PCA – Principal component analysis
SAE – Sparse Autoencoder
DAE – Denoising Autoencoder
CAE – Contractive Autoencoder
VAE – Variational Autoencoder
DTL – Deep Transfer Learning
DRL – Deep Reinforcement Learning
ОПМ – Обробка Природної Мови
РНМ – рекурентна нейронна мережа

МП – машинний переклад

НМП – нейронний машинний переклад

FLC – fixed-length encoding (кодування фіксованої довжини)

VLC – variable-length encoding (кодування змінної довжини)

LSTM – Long Short-Term Memory (довга короткочасна пам’ять)

ПК – пул кандидатів

РПК – розмір пулу кандидатів

EMD – Earth Mover’s Distance

ВСТУП

Глибинне навчання, галузь машинного навчання і штучного інтелекту, сьогодні вважається основною технологією сучасної Четвертої промислової революції. Завдяки своїм можливостям навчання з даних технологія глибинного навчання, яка походить від штучної нейронної мереж, стала гарячою темою в контексті обчислювальної техніки та широко використовується в різних сферах застосування, таких як охорона здоров'я, візуальне розпізнавання, текстова аналітика, кібербезпека та набагато більше. Однак побудова відповідної моделі є складним завданням через динамічну природу та варіації реальних проблем і даних. Крім того, відсутність основного розуміння перетворює методи глибинного навчання на машини чорного ящика, які перешкоджають розробці на стандартному рівні.

Останніми роками дослідники та практики обробки природної мови використали потужність сучасних штучних нейронних мереж із багатьма сприятливими результатами. За останні півтора десятиліття використання штучних нейронних мереж і глибинного навчання значно зросло в цій галузі. Це призвело до значного прогресу як в основних сферах обробки природної, так і в сферах, в яких воно безпосередньо застосовується для досягнення практичних і корисних цілей.

Глибинне навчання, на відміну від традиційного машинного навчання та алгоритмів інтелектуального аналізу даних, може створити представлення даних надзвичайно високого рівня з величезної кількості необроблених даних. Успішна техніка глибинного навчання повинна володіти релевантним моделюванням на основі даних залежно від характеристик необроблених даних. Потім складні алгоритми навчання потрібно навчити на основі зібраних даних і знань, пов'язаних із цільовою програмою, перш ніж система зможе допомогти з прийняттям розумних рішень.

1 ТЕХНОЛОГІЯ ГЛИБИННОГО НАВЧАННЯ, АНАЛІЗ ВЛАСТИВОСТЕЙ ТА ЗАЛЕЖНОСТЕЙ

Сьогодні технологія глибинного навчання вважається однією з гарячих тем у сфері машинного навчання, штучного інтелекту, також як і науки про дані та аналітику, завдяки її можливостям навчання на основі даних. Багато корпорацій, включаючи Google, Microsoft, Nokia тощо, активно вивчають її, оскільки вона може забезпечити значні результати в різних проблемах класифікації та регресії та наборах даних [1]. З точки зору робочої сфери, глибинне навчання розглядається як підмножина машинного навчання і штучного інтелекту, і, таким чином, глибинне навчання можна розглядати як функцію штучного інтелекту, яка імітує обробку даних людським мозком. Всесвітня популярність «глибинного навчання» зростає з кожним днем (на основі історичних даних, зібраних із Google Trends) [2]. Технологія глибинного навчання використовує кілька рівнів для представлення абстракцій даних для створення обчислювальних моделей. У той час як глибинне навчання займає багато часу для навчання моделі через велику кількість параметрів, воно займає короткий проміжок часу для запуску під час тестування, порівнюючи з іншими алгоритмами машинного навчання. Типова нейронна мережа в основному складається з багатьох простих, пов’язаних елементів обробки або процесорів, які називаються нейронами, кожен з яких генерує серію реальних активацій для цільового результату [3]. На рисунку 1.1 показано схематичне зображення математичної моделі штучного нейрона, тобто елемента обробки, з виділенням вхідних даних (Xi), ваги (w), зміщення (b), функції підсумовування (∑), функції активації (f) і відповідного вихідного сигналу (y).

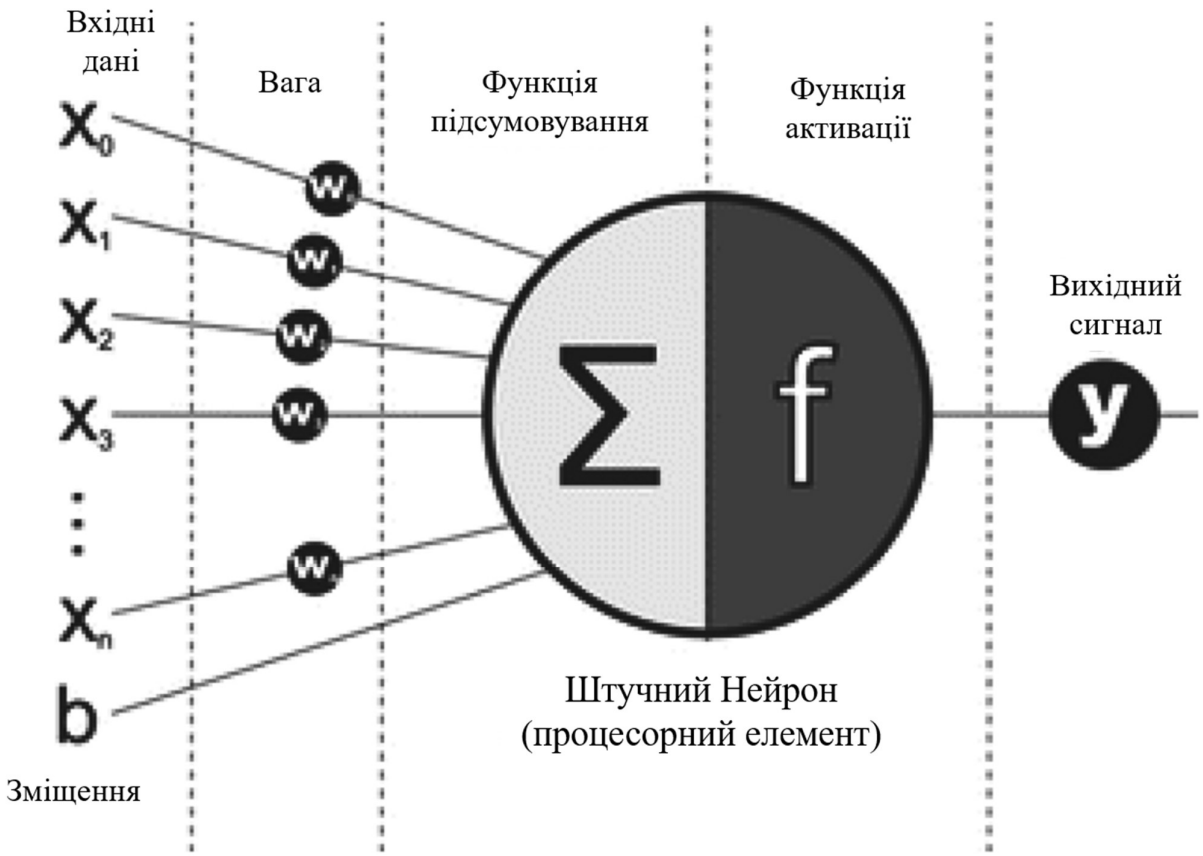

Рисунок 1.1 - Схематичне зображення математичної моделі штучного нейрона (процесорного елемента).

Технологія ГН на основі нейронної мережі зараз широко застосовується в багатьох галузях і областях досліджень, таких як охорона здоров’я, аналіз настроїв, обробка природної мови, візуальне розпізнавання, бізнес-аналітика, кібербезпека та багато інших.

Незважаючи на те, що моделі ГН успішно застосовуються в різних сферах застосування, згаданих вище, створення відповідної моделі глибинного навчання є складним завданням через динамічну природу та варіації проблем і даних реального світу. Крім того, ГН-моделі зазвичай вважаються машинами «чорної скриньки», які перешкоджають стандартній розробці досліджень і програм глибинного навчання. Таким чином, для чіткого розуміння ми представляємо структуроване та комплексне уявлення про методи ГН, враховуючи варіації проблем і завдань реального світу. Щоб досягти нашої мети, ми коротко

обговорюємо різні методи ГН і представляємо таксономію, беручи до уваги три основні категорії:

- глибинні мережі для контрольованого або дискримінаційного навчання, які використовуються для забезпечення дискримінаційної функції в контрольованому глибинному навчанні або програмах класифікації;
- глибинні мережі для неконтрольованого або генеративного навчання, які використовуються для характеристики кореляційних властивостей або особливостей високого порядку для аналізу або синтезу шаблонів, таким чином, можуть використовуватися як попередня обробка для керованого алгоритму;
- глибинні мережі для гібридного навчання, яке є інтеграцією як контрольованої, так і неконтрольованої моделі та відповідних інших.

Ми беремо до уваги такі категорії на основі природи та можливостей навчання різних методів ГН та того, як вони використовуються для вирішення проблем у реальних застосунках [4].

Підходи до глибинного навчання різко зросли з точки зору продуктивності в широкому діапазоні застосунків, враховуючи технології безпеки, зокрема як чудове рішення для розкриття складної архітектури у багатовимірних даних. Таким чином, методи ГН можуть відігравати ключову роль у створенні інтелектуальних систем, керованих даними, відповідно до сучасних потреб, завдяки їхнім відмінним можливостям навчання на основі історичних даних. Отже, ГН може змінити світ, а також повсякденне життя людей завдяки своїй потужності автоматизації та навчанню на досвіді. Таким чином, технологія ГН має відношення до штучного інтелекту, машинного навчання та науки про дані з передовою аналітикою, які є добре відомими областями інформатики, зокрема, сучасних інтелектуальних обчислень.

1.1 Позиція глибинного навчання в ШІ

Зараз штучний інтелект (ШІ), машинне навчання (МН) і глибинне навчання (ГН) — це три популярні терміни, які іноді використовуються як синоніми для

опису систем або програмного забезпечення, які поводяться розумно. На рис. 1.2 проілюстровано позицію глибинного навчання в порівнянні з машинним навчанням і штучним інтелектом.

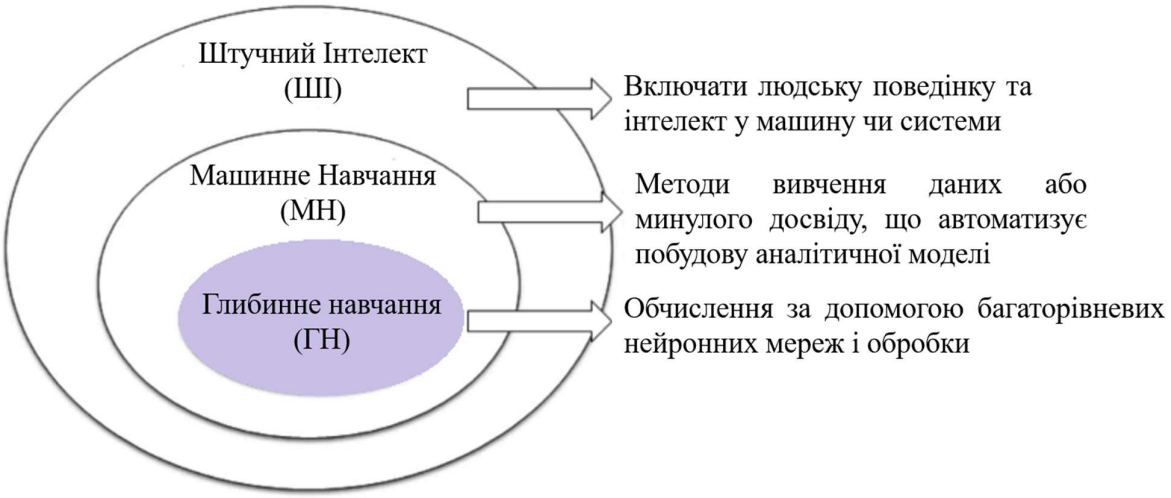

Рисунок 1.2 - Ілюстрація позиції глибинного навчання (ГН) у порівнянні з машинним навчанням (МН) і штучним інтелектом (ШІ).

Відповідно до рис. 2, ГН є частиною МН, а також частиною широкої області ШІ. Загалом штучний інтелект об’єднує людську поведінку та інтелект у машини чи системи [5], тоді як машинне навчання – це метод навчання на основі даних або досвіду, який автоматизує створення аналітичної моделі [4]. ГН також представляє методи навчання з даних, де обчислення виконуються за допомогою багаторівневих нейронних мереж і обробки. Термін «Глибинний» у методології глибинного навчання відноситься до концепції кількох рівнів або етапів, через які дані обробляються для побудови керованої даними моделі.

Таким чином, ГН можна розглядати як одну з основних технологій ШІ, передову для штучного інтелекту, яку можна використовувати для побудови інтелектуальних систем і автоматизації. Що ще важливіше, це висуває штучний інтелект на новий рівень, який називається «розумніший штучний інтелект». Оскільки ГН здатно навчатися з даних, глибинне навчання також має тісний зв’язок

із «Data Science» [6]. Як правило, наука про дані представляє собою весь процес пошуку сенсу або розуміння даних у конкретній проблемній області, де методи ГН можуть відігравати ключову роль для розширеної аналітики та інтелектуального прийняття рішень. Технологія ГН здатна змінити сучасний світ, зокрема, з точки зору потужного обчислювального механізму та сприяти автоматизації, що керується технологіями, розумним та інтелектуальним системам відповідно.

1.2 Властивості та залежності глибинного навчання

ГН-модель зазвичай має ті самі етапи обробки, що й моделювання машинного навчання. На рис. 1.3 показано робочий процес глибинного навчання для вирішення проблем реального світу, який складається з трьох етапів обробки, таких як розуміння та попередня обробка даних, побудова ГН-моделі та навчання, а також перевірка та інтерпретація.

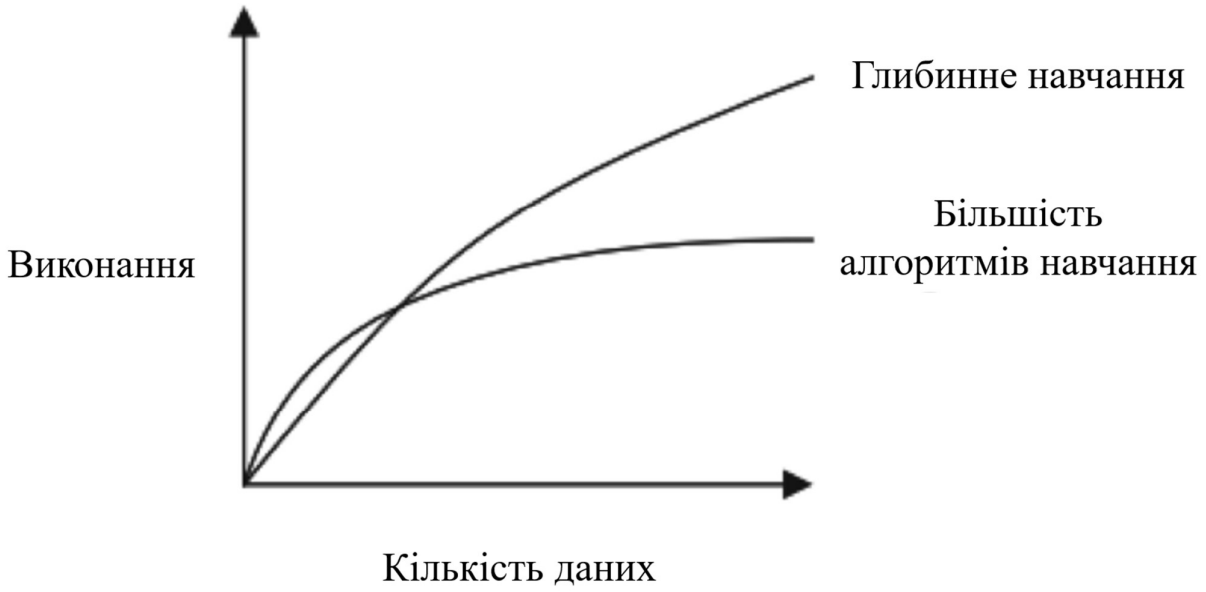

Рисунок 1.3 - Порівняння продуктивності між глибинним навчанням та іншими алгоритмами машинного навчання, де моделювання ГН на основі великих обсягів даних може підвищити продуктивність

Однак, на відміну від моделювання МН, виділення ознак у моделі ГН є автоматизованим, а не ручним. K-найближчий сусід, опорні векторні машини, дерево рішень, випадковий ліс, наївний байєсівський метод, лінійна регресія,

правила асоціації, кластеризація k-середніх – це деякі приклади методів машинного навчання, які зазвичай використовуються в різних областях застосування.

Ключові властивості та залежності методів DL, які необхідно взяти до уваги перед початком роботи над моделюванням DL для реальних додатків:

- Залежності від даних. Глибинне навчання зазвичай залежить від великої кількості даних для побудови моделі на основі даних для конкретної проблемної області. Причина полягає в тому, що коли обсяг даних невеликий, алгоритми глибинного навчання часто працюють погано [7]. Однак за таких обставин продуктивність стандартних алгоритмів машинного навчання буде покращена, якщо використовуватимуться вказані правила;
- Апаратні залежності. Алгоритми ГН вимагають великих обчислювальних операцій під час навчання моделі з великими наборами даних. Оскільки чим більші обчислення, тим більша перевага графічного процесора над центральним процесором, графічний процесор переважно використовується для ефективної оптимізації операцій. Таким чином, для належної роботи з глибоким навчанням необхідне обладнання GPU. Тому ГН більше покладається на високопродуктивні машини з графічним процесором, ніж на стандартні методи машинного навчання;
- Процес розробки функцій. Розробка функцій — це процес вилучення ознак (характеристик, властивостей і атрибутів) із необроблених даних за допомогою знань предметної області. Фундаментальною відмінністю між ГН та іншими методами машинного навчання є спроба отримати високорівневі характеристики безпосередньо з даних. Таким чином, ГН зменшує час і зусилля, необхідні для побудови екстрактора ознак для кожної проблеми;
- Навчання моделі та час виконання. Загалом навчання алгоритму глибинного навчання займає багато часу через велику кількість параметрів в алгоритмі ГН, таким чином, процес навчання моделі триває довше. Наприклад, для моделей ГН може знадобитися більше одного тижня, щоб завершити сеанс навчання, тоді як навчання з алгоритмами МН займає відносно мало часу, лише від секунд до годин. Під час тестування алгоритми глибокого навчання

потребують надзвичайно мало часу для запуску, порівнюючи з певними методами машинного навчання;

- Сприйняття чорної скриньки та можливість інтерпретації. Інтерпретація є важливим фактором при порівнянні ГН з МН. Важко пояснити, як був отриманий результат глибинного навчання, тобто «чорна скринька». З іншого боку, алгоритми машинного навчання, зокрема методи машинного навчання на основі правил, забезпечують чіткі логічні правила (ЯКЩО-ТОДІ) для прийняття рішень, які легко інтерпретуються людьми. Наприклад, машини, які навчаються методам, заснованим на правилах, де витягнуті правила є зрозумілими людині та їх легше інтерпретувати, оновлювати або видаляти відповідно до цільових програм.

Найсуттєвіша відмінність між глибинним навчанням і звичайним машинним навчанням полягає в тому, наскільки добре воно працює, коли дані зростають експоненціально. Ілюстрацію порівняння продуктивності між ГН і стандартними алгоритмами МН показано на рис. 1.3, де моделювання ГН може підвищити продуктивність із збільшенням обсягу даних. Таким чином, ГН-моделювання є надзвичайно корисним при роботі з великою кількістю даних через його здатність обробляти величезну кількість функцій для побудови ефективної моделі, керованої даними. З точки зору розробки та навчання моделей ГН, воно спирається на розпаралелені матриці та тензорні операції, а також обчислювальні градієнти та оптимізацію. Кілька бібліотек і ресурсів ГН, таких як PyTorch з API високого рівня під назвою Lightning і TensorFlow, який також пропонує Keras як API високого рівня, пропонують ці основні утиліти, включаючи багато попередньо навчених моделей, а також багато інших необхідних функцій для реалізації та побудови моделі ГН.

1.3 Методи та застосунки глибинного навчання

У цьому розділі ми розглядаємо різні типи методів глибинної нейронної мережі, які зазвичай розглядають кілька рівнів етапів обробки інформації в ієрархічних структурах для навчання. Типова глибока нейронна мережа містить

кілька прихованих рівнів, включаючи вхідний і вихідний рівні. На рис. 1.4 показано загальну структуру глибинної нейронної мережі (прихований рівень =N і N ≥ 2) у порівнянні з неглибинною мережею (прихований рівень =1).

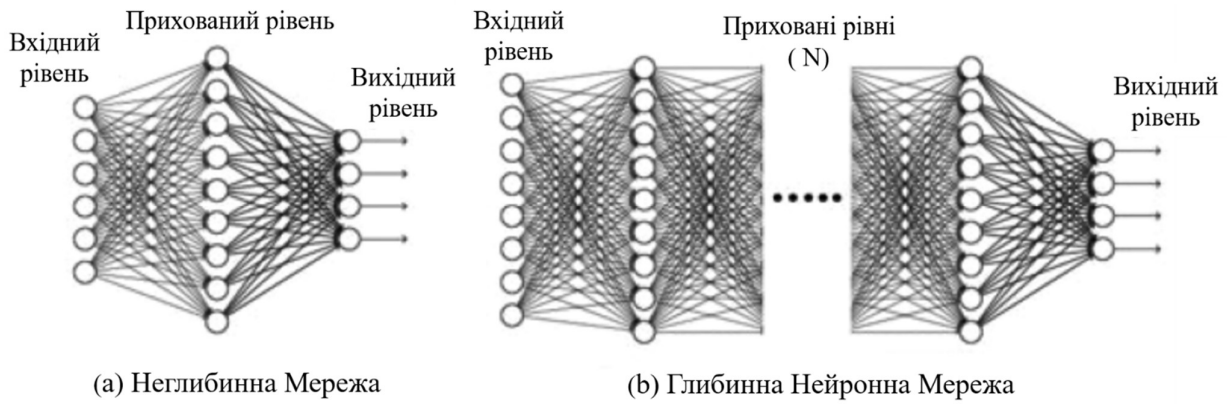

Рисунок 1.4 - Загальна архітектура (a) неглибинної мережі з одним прихованим рівнем і (b) глибинної нейронної мережі з кількома прихованими рівнями

Однак, перш ніж вивчати деталі методів DL, корисно переглянути різні типи навчальних завдань, наприклад:

- Під наглядом: підхід, орієнтований на завдання, який використовує позначені навчальні дані;
- Неконтрольований: керований даними процес, який аналізує непозначені набори даних;
- Напівконтрольований: гібридизація як контрольованих, так і неконтрольованих методів;
- Підкріплення: підхід, орієнтований на середовище.

2 АНАЛІЗ МЕТОДІВ ТА ЗАСТОСУНКІВ ТЕХНОЛОГІЇ ГЛИБИННОГО НАВЧАННЯ

Представимо схему (рис. 2.1), де методи глибинного навчання поділені на три основні категорії: глибинні мережі для контрольованого або дискримінаційного навчання; глибинні мережі для неконтрольованого або генеративного навчання; глибинні мережі для гібридного навчання, що поєднують обидва та відповідні інші.

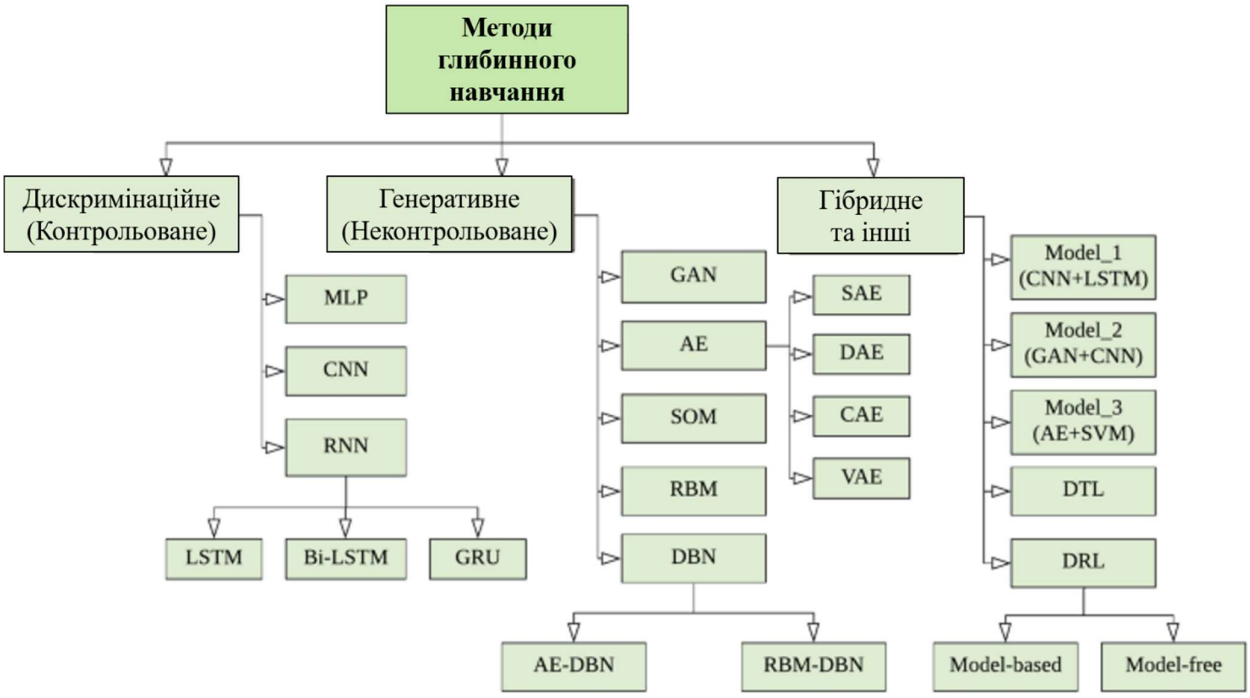

Рисунок 2.1 – Методи глибинного навчання

Далі коротко роздивимося кожен із цих методів, які можна використовувати для вирішення реальних проблем у різних сферах застосування відповідно до їхніх можливостей навчання.

2.1 Глибинні мережі для контрольованого або дискримінаційного навчання

Ця категорія методів ГН використовується для забезпечення розрізнювальної функції в контрольованих або класифікаційних програмах. Дискримінаційні глибинні архітектури, як правило, розроблені, щоб надати дискримінаційну силу для класифікації шаблонів шляхом опису апостеріорних розподілів класів, обумовлених видимими даними [8]. Дискримінаційні архітектури в основному

включають багаторівневий персептрон (БРП, MLP), згорткові нейронні мережі (ЗНМ, CNN або ConvNet), рекурентні нейронні мережі (РНМ, RNN) разом із їхніми варіантами.

2.1.1 Multi-Layer Perceptron

Багаторівневий персептрон є різновидом штучної нейронної мережі (ШНМ, ANN) прямого зв'язку. Він також відомий як фундаментальна архітектура глибинних нейронних мереж (ГНМ, DNN) або глибинного навчання. Типова MLP — це повністю підключена мережа, яка складається з вхідного рівня, який отримує вхідні дані, вихідного рівня, який приймає рішення або передбачає прогноз щодо вхідного сигналу, і одного або кількох прихованих рівнів між цими двома, які вважаються обчислювальним механізмом мережі. Вихід мережі MLP визначається за допомогою різноманітних функцій активації, також відомих як функції передачі, таких як ReLU (Rectified Linear Unit), Tanh, Sigmoid і Softmax. Для навчання MLP використовується найбільш широко використовуваний алгоритм «Зворотне поширення», методика навчання під наглядом, яка також відома як найпростіший будівельний блок нейронної мережі. У процесі навчання застосовуються різні підходи до оптимізації, такі як стохастичний градієнтний спуск (СГС, SGD), BFGS з обмеженою пам'яттю (L-BFGS) і оцінка адаптивного моменту (Adam). MLP вимагає налаштування кількох гіперпараметрів, таких як кількість прихованих рівнів, нейронів і ітерацій, що може зробити вирішення складної моделі обчислювально дорогим. Однак завдяки частковій відповідності MLP пропонує перевагу вивчення нелінійних моделей у режимі реального часу або онлайн.

2.1.2 Convolutional Neural Network

Згорткова нейронна мережа — це популярна дискримінаційна архітектура глибинного навчання, яка вчиться безпосередньо на вхідних даних без необхідності вилучення функцій людиною. На рисунку 2.2 показаний приклад CNN, що включає кілька згорток і рівні об'єднання.

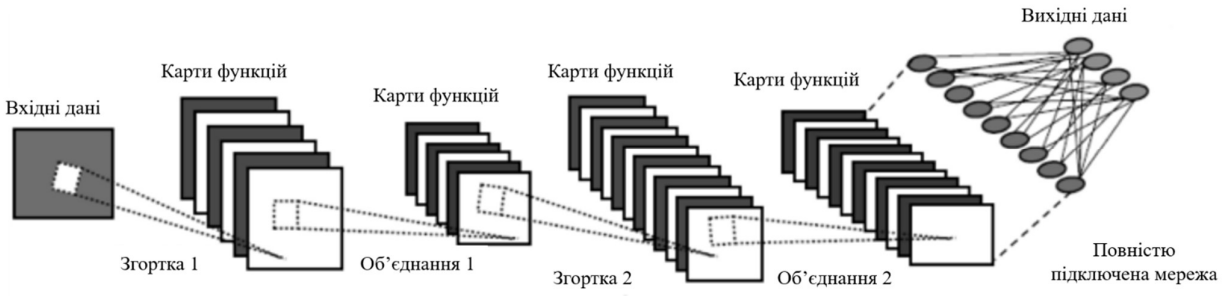

Рисунок 2.2 - Приклад згорткової нейронної мережі, що включає кілька рівнів згортки та об'єднання

Як наслідок, CNN покращує дизайн традиційних ШНМ, таких як упорядковані мережі MLP. Кожен рівень у CNN враховує оптимальні параметри для значущого результату, а також зменшує складність моделі. CNN також використовує «випадок», який може вирішити проблему надмірної підгонки, яка може виникнути в традиційній мережі.

CNN спеціально призначені для роботи з різними двовимірними формами і, таким чином, широко використовуються у візуальному розпізнаванні, аналізі медичних зображень, сегментації зображень, обробці природної мови та багато іншого. Можливість автоматичного виявлення основних функцій на вхідних даних без втручання людини робить її потужнішою, ніж традиційна мережа. Існує кілька варіантів CNN, які включають групу візуальної геометрії (ГВГ, VGG), AlexNet, Xception, Inception, ResNet тощо, які можна використовувати в різних застосунках, відповідно до їхніх можливостей навчання.

2.1.3 Рекурентна нейронна мережа та її варіації

Рекурентна нейронна мережа (RNN) — це ще одна популярна нейронна мережа, яка використовує послідовні або часові дані та передає вихідні дані попереднього кроку як вхідні дані для поточного етапу. Подібно до прямого зв'язку та CNN, рекурентні мережі навчаються на вхідних даних навчання, однак відрізняються своєю «пам'яттю», яка дозволяє їм впливати на поточні вхідні та вихідні дані за допомогою інформації з попередніх вхідних даних. На відміну від типового DNN, який передбачає, що входи та виходи незалежні один від одного, вихід RNN залежить від попередніх елементів у послідовності. Однак стандартні

рекурентні мережі мають проблему зникнення градієнтів, що ускладнює вивчення довгих послідовностей даних. Далі приведено кілька популярних варіантів рекурентної мережі, яка мінімізує проблеми та добре працює в багатьох реальних сферах застосування.

- Довга короткочасна пам’ять (LSTM) Це популярна форма архітектури RNN, яка використовує спеціальні блоки для вирішення проблеми зникнення градієнта. Комірка пам'яті в блоці LSTM може зберігати дані протягом тривалого часу, а потік інформації в комірку та з неї керується трьома шлюзами. Наприклад, «Forget Gate» визначає, яка інформація з комірки попереднього стану буде запам’ятовуватися, а яка інформація буде видалена, що більше не є корисною, тоді як «Input Gate» визначає, яка інформація повинна вводитися в стан комірки, а «Output Gate» визначає та контролює виходи. Оскільки вона вирішує питання навчання рекурентної мережі, мережа LSTM вважається однією з найуспішніших RNN.
- Двонаправлені RNN/LSTM. Двонаправлені RNN з’єднують два приховані рівні, які йдуть у протилежних напрямках, до одного виходу, дозволяючи їм приймати дані як з минулого, так і з майбутнього. Двонаправлені RNN, на відміну від традиційних рекурентних мереж, навчені прогнозувати як позитивні, так і негативні напрямки часу одночасно. Двонаправлена LSTM, часто відома як BiLSTM, є розширенням стандартної LSTM, яка може підвищити продуктивність моделі щодо питань класифікації послідовностей. Це модель обробки послідовності, що складається з двох LSTM: одна передає вхідні дані вперед, а інша – назад. Зокрема, двонаправлена LSTM є популярним вибором у задачах обробки природної мови.
- Шлюзовані рекурентні блоки (GRU). Шлюзований рекурентний блок — ще один популярний варіант рекурентної мережі, який використовує методи шлюзів для контролю та керування потоком інформації між клітинами нейронної мережі. GRU схожий на LSTM, але має менше параметрів, оскільки має шлюз скидання та шлюз оновлення, але не має вихідного шлюзу, як показано на рис. 2.3. Таким чином, ключова відмінність між GRU

та LSTM полягає в тому, що GRU має два шлюзи (шлюзи скидання та оновлення), тоді як LSTM має три шлюзи (а саме шлюзи входу, виходу та шлюзи забуття). Структура GRU дозволяє адаптивно фіксувати залежності від великих послідовностей даних, не відкидаючи інформацію з попередніх частин послідовності. Таким чином, GRU є дещо більш спрощеним варіантом, який часто пропонує порівнянну продуктивність і значно швидший для обчислень.

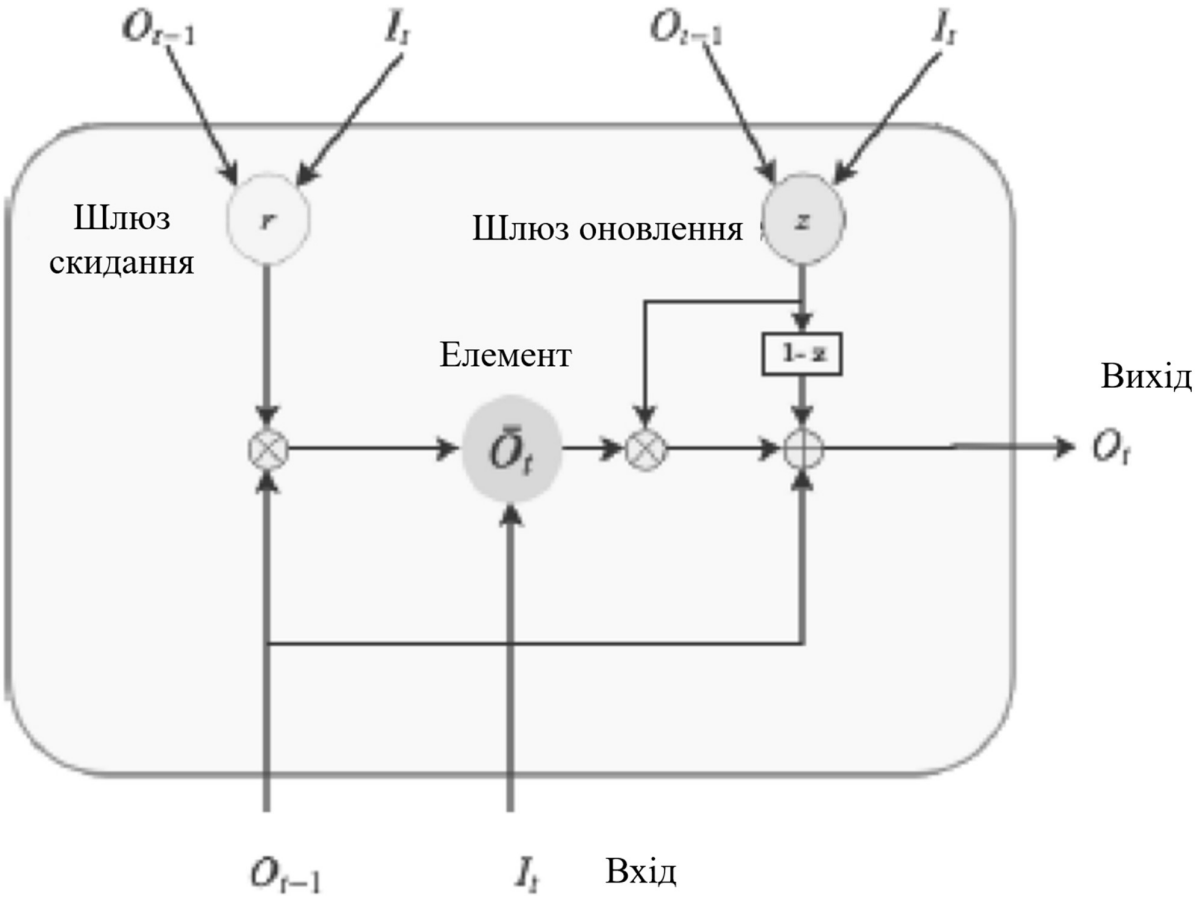

Рисунок 2.3 - Базова структура елементу шлюзованого рекурентного блоку, що складається з шлюзів скидання та оновлення

Незважаючи на те, що було показано, що GRU демонструють кращу продуктивність на певних менших і менш частих наборах даних, усі варіанти RNN довели свою ефективність при отриманні результату.

Загалом, основна властивість рекурентної мережі полягає в тому, що вона має принаймні одне з’єднання зі зворотним зв’язком, що дозволяє циклічність

активації. Це дозволяє мережам здійснювати часову обробку та навчання послідовності, наприклад розпізнавання або відтворення послідовності, часову асоціацію або передбачення тощо. Нижче наведено деякі популярні області застосування повторюваних мереж, такі як проблеми передбачення, машинний переклад, обробка природної мови, підсумовування тексту, розпізнавання мовлення та багато іншого.

2.2 Глибинні мережі для неконтрольованого або генеративного навчання

Ця категорія методів ГН зазвичай використовується для характеристики кореляційних властивостей або особливостей високого порядку для аналізу або синтезу шаблонів, а також спільних статистичних розподілів видимих даних і пов’язаних з ними класів. Ключова ідея генеративних глибинних архітектур полягає в тому, що під час процесу навчання точна контрольна інформація, така як мітки цільового класу, не має значення. Як наслідок, методи цієї категорії в основному застосовуються для неконтрольованого навчання, оскільки методи зазвичай використовуються для вивчення ознак або генерації та представлення даних. Таким чином, генеративне моделювання також може використовуватися як попередня обробка для контрольованих навчальних завдань, що забезпечує точність дискримінаційної моделі. Методи глибокої нейронної мережі, які зазвичай використовуються для неконтрольованого або генеративного навчання — це Generative Adversarial Network (GAN), Auto-Encoder (AE), Restricted Boltzmann Machine (RBM), Self-Organizing Map (SOM) і Deep Belief Network (DBN) разом із їхніми варіантами.

2.2.1 Generative Adversarial Network

Generative Adversarial Network, розроблена Іаном Гудфеллоу [9], є типом архітектури нейронної мережі для генеративного моделювання для створення нових вірогідних зразків на вимогу. Це передбачає автоматичне виявлення та вивчення закономірностей у вхідних даних, щоб модель могла використовуватися для створення або виведення нових прикладів із вихідного набору даних. Як показано на рис. 2.4, GAN складаються з двох нейронних мереж: генератора G, який створює нові дані з властивостями, подібними до вихідних даних, і

дискримінатора D, який передбачає ймовірність того, що наступна вибірка буде взята з фактичних даних, а не з даних забезпечується генератором.

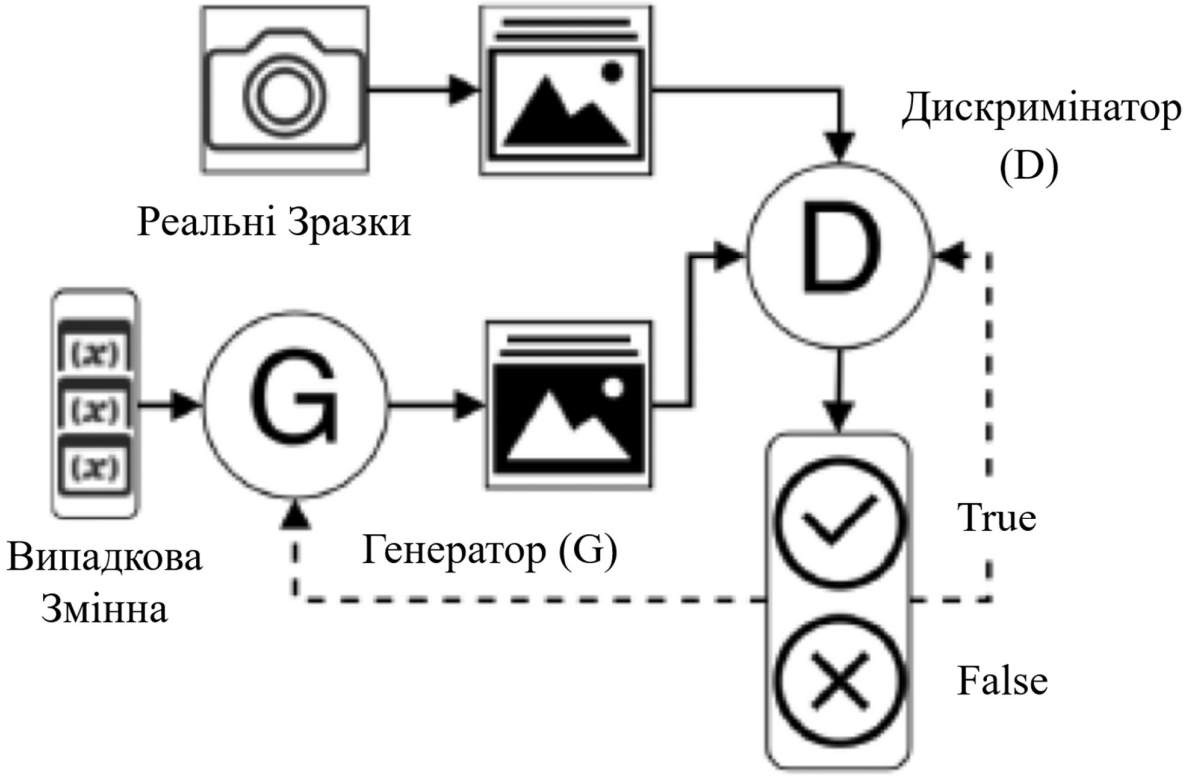

Рисунок 2.4 - Схематична структура стандартної Generative Adversarial Network

Таким чином, у моделюванні GAN і генератор, і дискримінатор навчені конкурувати один з одним. У той час як генератор намагається обдурити та заплутати дискримінатор, створюючи більш реалістичні дані, дискримінатор намагається відрізнити справжні дані від підроблених даних, створених генератором.

Загалом розгортання мережі GAN розроблено для неконтрольованих завдань навчання, але воно також виявилося кращим рішенням для напівконтрольованого навчання та навчання з підкріпленням також, залежно від завдання. GAN також використовуються в найсучасніших дослідженнях переносу навчання для забезпечення вирівнювання простору прихованих ознак. Інверсні моделі, такі як двонаправлена GAN (BiGAN), також можуть вивчати відображення даних у прихований простір, подібно до того, як стандартна модель GAN вивчає відображення від прихованого простору до розподілу даних. Потенційні сфери застосування мереж GAN – це охорона здоров’я, аналіз зображень, доповнення

даних, генерація відео, генерація голосу, пандемії, контроль трафіку, кібербезпека та багато інших, які швидко зростають. Загалом GAN зарекомендували себе як всеосяжний домен незалежного розширення даних і як рішення проблем, що вимагають генеративного рішення.

2.2.2 Auto-Encoder та його варіації

Автокодер [10] є популярною технікою неконтрольованого навчання, у якій нейронні мережі використовуються для вивчення представлень. Як правило, автокодери використовуються для роботи з даними великої розмірності, а зменшення розмірності пояснює, як представляється набір даних. Кодер, код і декодер — це три частини автокодувальника. Кодер стискає вхідні дані та генерує код, який згодом використовує декодер для реконструкції вхідних даних. AE нещодавно використовувалися для вивчення генеративних моделей даних. Автокодер широко використовується в багатьох неконтрольованих навчальних завданнях, наприклад, зменшення розмірності, виділення ознак, ефективне кодування, генеративне моделювання, усунення шумів, виявлення аномалій або викидів тощо. Аналіз головних компонент (PCA), який також використовується для зменшення розмірності величезних наборів даних, по суті схожий на однорівневий AE з лінійною функцією активації. Регуляризовані автокодери, такі як розріджені, знешумні та скорочувальні, корисні для вивчення представлень для пізніших завдань класифікації, тоді як варіаційні автокодери можуть використовуватися як генеративні моделі:

- Розріджений автокодер (SAE). Розріджений автокодер має штраф за розрідженість на рівні кодування як частину його вимог до навчання. SAE може мати більше прихованих одиниць, ніж вхідних даних, але лише невеликій кількості прихованих одиниць дозволяється бути активними одночасно, що призводить до розрідженої моделі. На рисунку 2.5 показана схематична структура розрідженого автокодувальника з кількома активними блоками в прихованому рівні. Таким чином, ця модель зобов’язана реагувати на унікальні статистичні характеристики навчальних даних, дотримуючись своїх обмежень.

- Автокодер з усуненням шумів (DAE). Автокодер із усуненням шумів є варіантом базового автокодувальника, який намагається покращити представлення (для вилучення корисних функцій) шляхом зміни критерію реконструкції, і таким чином зменшує ризик вивчення функції ідентифікації. Іншими словами, він отримує пошкоджену точку даних як вхідні дані та навчається відновлювати неспотворені вхідні дані як вихідні дані шляхом мінімізації середньої помилки реконструкції над навчальними даними, тобто очищення пошкоджених вхідних даних або усунення шумів. Таким чином, у контексті обчислень DAE можна розглядати як дуже потужні фільтри, які можна використовувати для автоматичної попередньої обробки. Наприклад, для автоматичної попередньої обробки зображення можна використовувати автоматичний кодер із шумозаглушенням, що підвищує його якість для точності розпізнавання.
- Контрактивний автокодер (CAE). Ідея контрактивного автокодера, полягає в тому, щоб зробити автокодери стійкими до невеликих змін у навчальному наборі даних. У своїй цільовій функції CAE містить явний регуляризатор, який змушує модель вивчати кодування, стійке до невеликих змін у вхідних значеннях. Як наслідок, чутливість вивченого представлення до тренувальних даних знижується. Хоча DAE заохочують надійність реконструкції, як обговорювалося вище, CAE заохочують надійність представлення.
- Варіаційний автокодер (VAE). Варіаційний автокодер має принципово унікальну властивість, яка відрізняє його від класичного автокодеру, розглянутого вище, що робить його настільки ефективним для генеративного моделювання. VAE, на відміну від традиційних автокодерів, які відображають вхідні дані на прихований вектор, відображають вхідні дані в параметрах розподілу ймовірностей, таких як середнє значення та дисперсія розподілу Гауса. VAE припускає, що вихідні дані мають базовий розподіл ймовірностей, а потім намагається виявити параметри розподілу. Хоча цей підхід спочатку був розроблений для неконтрольованого навчання, його

використання було продемонстровано в інших областях, таких як напівконтрольоване навчання і контрольоване навчання.

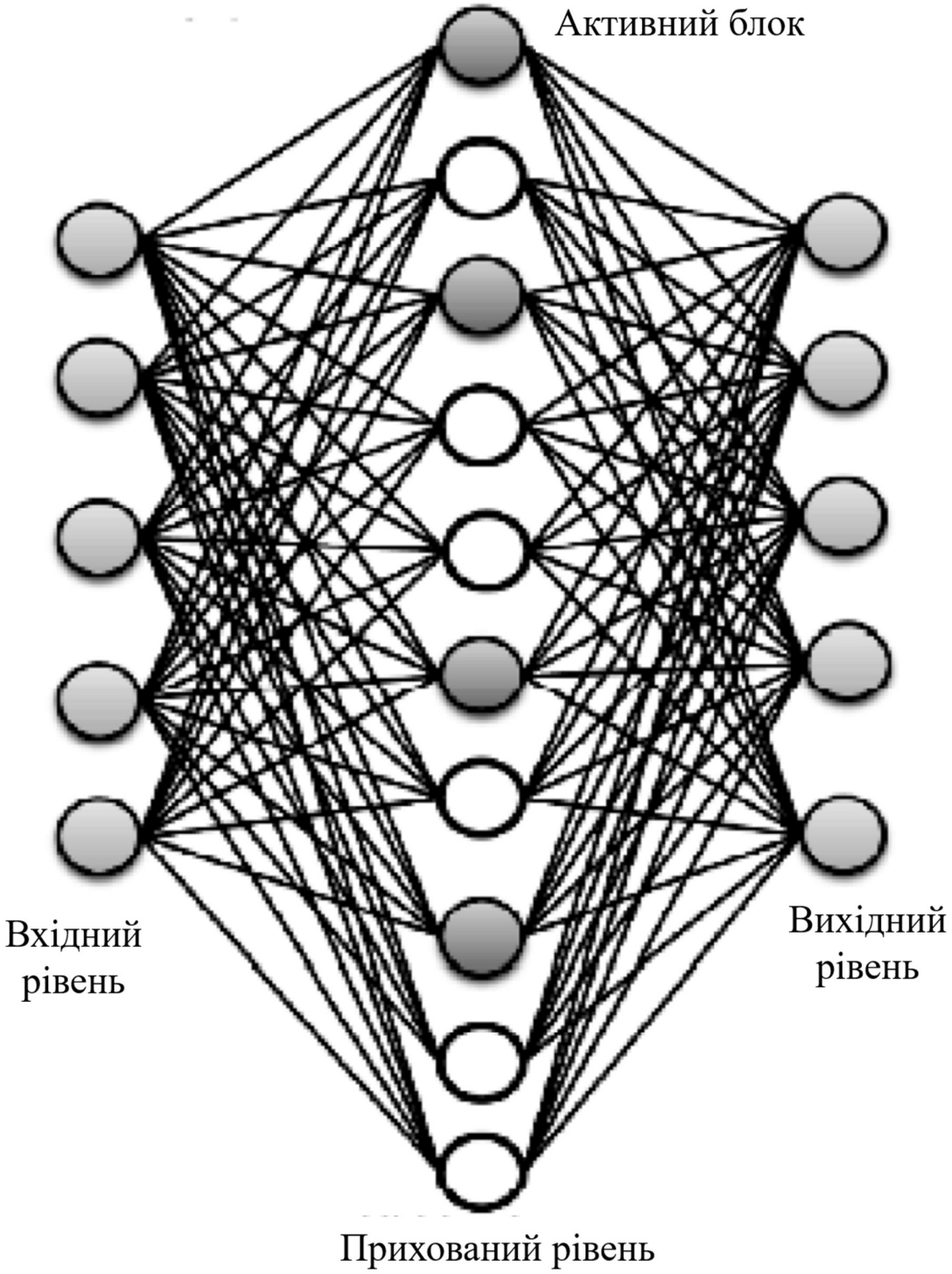

Рисунок 2.5 - Схематична структура розрідженого автокодера з кількома активними блоками (заповнене коло) у прихованому рівні

Хоча попередня концепція АЕ, як правило, була для зменшення розмірності або вивчення ознак, згаданих вище, останнім часом АЕ були виведені на передній план генеративного моделювання, навіть генеративна змагальна мережа є одним із популярних методів у цій області. АЕ ефективно використовуються в різних сферах, включаючи охорону здоров'я, комп'ютерний зір, розпізнавання мови, кібербезпеку, обробку природної мови та багато іншого. Загалом можна зробити висновок, що автоматичний кодувальник і його варіанти можуть відігравати значну роль як неконтрольоване навчання функцій з архітектурою нейронної мережі.

2.2.3 Self-Organizing Map

Самоорганізуюча карта [11] є іншою формою техніки неконтрольованого навчання для створення низьковимірного (зазвичай двовимірного) представлення набору даних з більшою вимірністю, зберігаючи топологічну структуру даних. SOM також відомий як алгоритм зменшення розмірності на основі нейронної мережі, який зазвичай використовується для кластеризації. SOM адаптується до топологічної форми набору даних, постійно переміщаючи свої нейрони ближче до точок даних, що дозволяє нам візуалізувати величезні набори даних і знаходити ймовірні кластери. Перший рівень SOM є вхідним, а другий – вихідним або картою об'єктів. На відміну від інших нейронних мереж, які використовують навчання з виправленням помилок, таких як зворотне поширення з градієнтним спуском, SOM використовують конкурентне навчання, яке використовує функцію сусідства для збереження топологічних особливостей вхідного простору. SOM широко використовується в різноманітних додатках, включаючи ідентифікацію шаблонів, діагностику стану здоров'я чи медичну діагностику, виявлення аномалій та виявлення атак вірусів чи хробаків. Основна перевага використання SOM полягає в тому, що це може полегшити візуалізацію та аналіз високорозмірних даних, щоб зрозуміти закономірності. Зменшення розмірності та кластеризація сітки дозволяє легко спостерігати схожість у даних. У результаті SOM можуть відігравати важливу роль у розробці керованої даними ефективної моделі для конкретної проблемної області залежно від характеристик даних.

2.2.4 Restricted Boltzmann Machine

Обмежена машина Больцмана [12] також є генеративною стохастичною нейронною мережею, здатною навчатися розподілу ймовірностей за своїми входами. Машини Больцмана зазвичай складаються з видимих і прихованих вузлів, і кожен вузол з'єднаний з кожним іншим вузлом, що допомагає нам зрозуміти порушення, вивчаючи, як система працює в нормальних умовах. RBM є підмножиною машин Больцмана, які мають обмеження на кількість з'єднань між видимими і прихованими рівнями. Це обмеження дозволяє алгоритмам навчання, таким як алгоритм контрастної дивергенції на основі градієнта, бути більш ефективним, ніж для машин Больцмана в цілому. RBM знайшли застосування у зменшенні розмірності, класифікації, регресії, спільному фільтруванні, навчанні функцій, тематичному моделюванні та багатьох інших. У сфері моделювання глибокого навчання їх можна навчати як під наглядом, так і без нагляду, залежно від завдання. Загалом, RBM можуть автоматично розпізнавати закономірності в даних і розробляти імовірнісні або стохастичні моделі, які використовуються для вибору або вилучення ознак, а також для формування мережі глибоких переконань.

2.2.5 Deep Belief Network

Мережа глибокої віри [13] — це багаторівнева генеративна графічна модель стеку кількох окремих неконтрольованих мереж, таких як AE або RBM, які використовують прихований рівень кожної мережі як вхід для наступного рівня, тобто з'єднані послідовно. Таким чином, ми можемо розділити DBN на:

- AE-DBN, який відомий як стекований AE;
- (ii) RBM-DBN, який відомий як стековий RBM,

де AE-DBN складається з автокодерів, а RBM-DBN складається з обмеженої машини Больцмана, про які йшлося раніше. Кінцева мета полягає в тому, щоб розробити швидшу безконтрольну техніку навчання для кожної підмережі, яка залежить від контрастної дивергенції. DBN може фіксувати ієрархічне представлення вхідних даних на основі їх глибинної структури. Основна ідея DBN полягає в тому, щоб навчити неконтрольовані нейронні мережі прямого зв'язку з

даними без міток перед тонким налаштуванням мережі з мітками. Одна з найважливіших переваг DBN, на відміну від типових дрібних мереж навчання, полягає в тому, що вона дозволяє виявляти глибокі шаблони, що дає змогу розуміти та фіксувати глибоку різницю між нормальними та помилковими даними. Безперервний DBN — це просто розширення стандартного DBN, який дозволяє безперервний діапазон десяткових знаків замість двійкових даних. Загалом, модель DBN може відігравати ключову роль у широкому діапазоні додатків високовимірних даних завдяки своїм потужним можливостям вилучення ознак і класифікації та стати однією з важливих тем у галузі нейронних мереж.

Таким чином, методи генеративного навчання, розглянуті вище, зазвичай дозволяють створити нове представлення даних за допомогою дослідницького аналізу. У результаті ці глибокі генеративні мережі можна використовувати як попередню обробку для контрольованих або дискримінаційних завдань навчання, а також для забезпечення точності моделі, де неконтрольоване навчання представлення може дозволити покращити узагальнення класифікатора.

2.3 Глибинні мережі для гібридного навчання

На додаток до розглянутих вище категорій глибокого навчання популярними є гібридні глибинні мережі та кілька інших підходів, таких як глибинне навчання з перенесенням (DTL) і глибинне навчання з підкріпленням (DRL), які обговорюються далі.

2.3.1 Гібридні глибинні нейронні мережі

Генеративні моделі є адаптивними, вони здатні навчатися як на позначених, так і на немаркованих даних. Дискримінаційні моделі, з іншого боку, не можуть навчатися з немаркованих даних, але перевершують своїх генеративних аналогів у контрольованих завданнях. Структура для навчання як глибоких генеративних, так і дискримінаційних моделей одночасно може користуватися перевагами обох моделей, що мотивує гібридні мережі.

Гібридні моделі глибокого навчання зазвичай складаються з кількох (двох або більше) моделей глибокого базового навчання, де базовою моделлю є дискримінаційна або генеративна модель глибокого навчання, про яку говорилося

раніше. На основі інтеграції різних базових генеративних або дискримінаційних моделей наведені нижче три категорії гібридних моделей глибокого навчання можуть бути корисними для вирішення проблем реального світу. Це такі:

- Гібридна модель 1: інтеграція різних генеративних або дискримінаційних моделей для отримання більш значущих і надійних функцій. Прикладами можуть бути CNN+LSTM, AE+GAN тощо;
- Гібридна модель 2: інтеграція генеративної моделі з наступною дискримінаційною моделлю. Прикладами можуть бути DBN+MLP, GAN+CNN, AE+CNN тощо;
- Гібридна модель 3: інтеграція генеративної або дискримінаційної моделі з подальшим класифікатором неглибокого навчання. Прикладами можуть бути AE+SVM, CNN+SVM тощо.

Таким чином, у широкому розумінні можна зробити висновок, що гібридні моделі можуть бути орієнтованими на класифікацію або не класифікувати залежно від цільового використання. Однак більшість досліджень, пов’язаних із гібридним навчанням у сфері глибокого навчання, зосереджені на класифікації або контрольованих навчальних завданнях. Неконтрольовані генеративні моделі зі значущими представленнями використовуються для покращення дискримінаційних моделей. Генеративні моделі з корисним представленням можуть надавати більш інформативні та маловимірні ознаки для розрізнення, а також можуть дозволити покращити якість і кількість навчальних даних, надаючи додаткову інформацію для класифікації.

2.3.2 Deep Transfer Learning

Трансферне навчання — це техніка ефективного використання попередньо отриманих модельних знань для вирішення нового завдання з мінімальним навчанням або тонким налаштуванням. У порівнянні з типовими методами машинного навчання, ГН використовує велику кількість навчальних даних. Як наслідок, потреба у значному обсязі мічених даних є суттєвою перешкодою для вирішення деяких важливих завдань, пов’язаних із предметною областю, зокрема

в медичному секторі, де створення великомасштабних високоякісних анотованих наборів медичних або медичних даних є водночас складним. і дорого. Крім того, стандартна модель ГН вимагає багато обчислювальних ресурсів, таких як сервер із підтримкою GPU, хоча дослідники наполегливо працюють над її вдосконаленням. Як наслідок, DTL, метод навчання передачі на основі ГН, може бути корисним для вирішення цієї проблеми. На рисунку 2.6 показана загальна структура процесу навчання передачі, де знання з попередньо навченої моделі переносяться в нову модель ГН. Зараз він особливо популярний у глибинному навчанні, оскільки дозволяє навчати глибокі нейронні мережі з дуже невеликою кількістю даних.

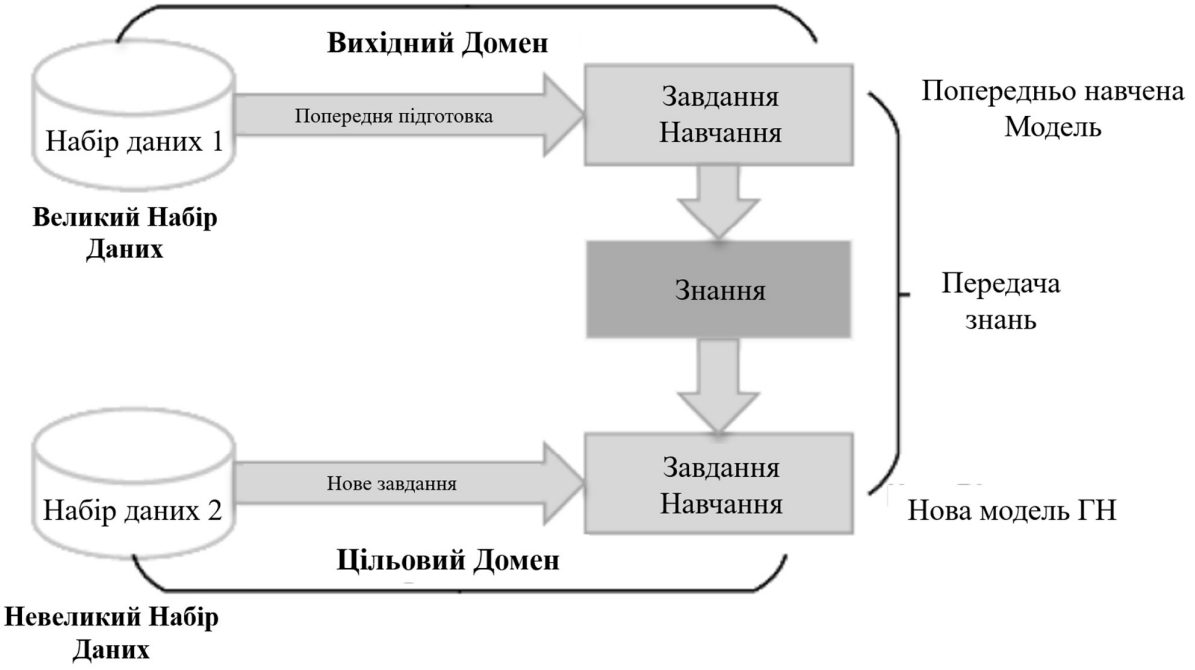

Рисунок 2.6 - Загальна структура процесу трансферного навчання, де знання з попередньо навченої моделі переносяться в нову модель глибинного навчання

Трансферне навчання — це двоетапний підхід до навчання ГН-моделі, який складається з етапу попереднього навчання та етапу тонкого налаштування, під час якого модель навчається для цільового завдання. Оскільки глибинні нейронні мережі набули популярності в різних сферах, було представлено велику кількість методів DTL, що робить вкрай важливим їх класифікацію та узагальнення. На основі методів, які використовуються в літературі, DTL можна класифікувати на чотири категорії:

- глибинне навчання передачі на основі екземплярів, яке використовує екземпляри у вихідному домені відповідною вагою;
- глибинне навчання на основі відображення, яке відображає екземпляри з двох доменів у новий простір даних із кращою подібністю;
- мережеве глибинне навчання на основі передачі, яке повторно використовує частину попередньо навченої мережі у вихідному домені;
- глибинне навчання на основі змагальності, яке використовує змагальну технологію для пошуку функцій, які можна передавати, які обидва підходять для двох доменів.

Завдяки своїй високій ефективності та практичності, змагальність на основі глибинного трансферу навчання набула популярності в останні роки. Трансферне навчання також можна класифікувати на індуктивне, трансдуктивне та неконтрольоване трансферне навчання залежно від обставин між вихідною та цільовою сферами та діяльністю. У той час як більшість сучасних досліджень зосереджені на контрольованому навчанні, те, як глибинні нейронні мережі можуть передавати знання в неконтрольованому або напівконтрольованому навчанні, може викликати додатковий інтерес у майбутньому. Методи DTL корисні в різних сферах, включаючи обробку природної мови, класифікацію настроїв, візуальне розпізнавання, розпізнавання мови, фільтрацію спаму та інші відповідні.

2.3.3 Deep Reinforcement Learning

Навчання з підкріпленням використовує інший підхід до вирішення проблеми послідовного прийняття рішень, ніж інші підходи, які ми обговорювали досі. Поняття середовища та агенту часто вводяться спочатку під час навчання з підкріпленням. Агент може виконувати низку дій у середовищі, кожна з яких впливає на стан середовища та може призвести до можливих винагород (зворотного зв’язку) – «позитивний» за хороші послідовності дій, які призводять до «хорошого» стану, і «негативний» для неправильної послідовності дій, які призводять до «поганого» стану. Мета навчання з підкріпленням полягає в тому,

щоб засвоїти хороші послідовності дій через взаємодію з навколишнім середовищем, що зазвичай називають політикою.

Глибоке навчання з підкріпленням об'єднує нейронні мережі з архітектурою навчання з підкріпленням, щоб дозволити агентам навчатися відповідним діям у віртуальному середовищі, як показано на рис. 2.7.

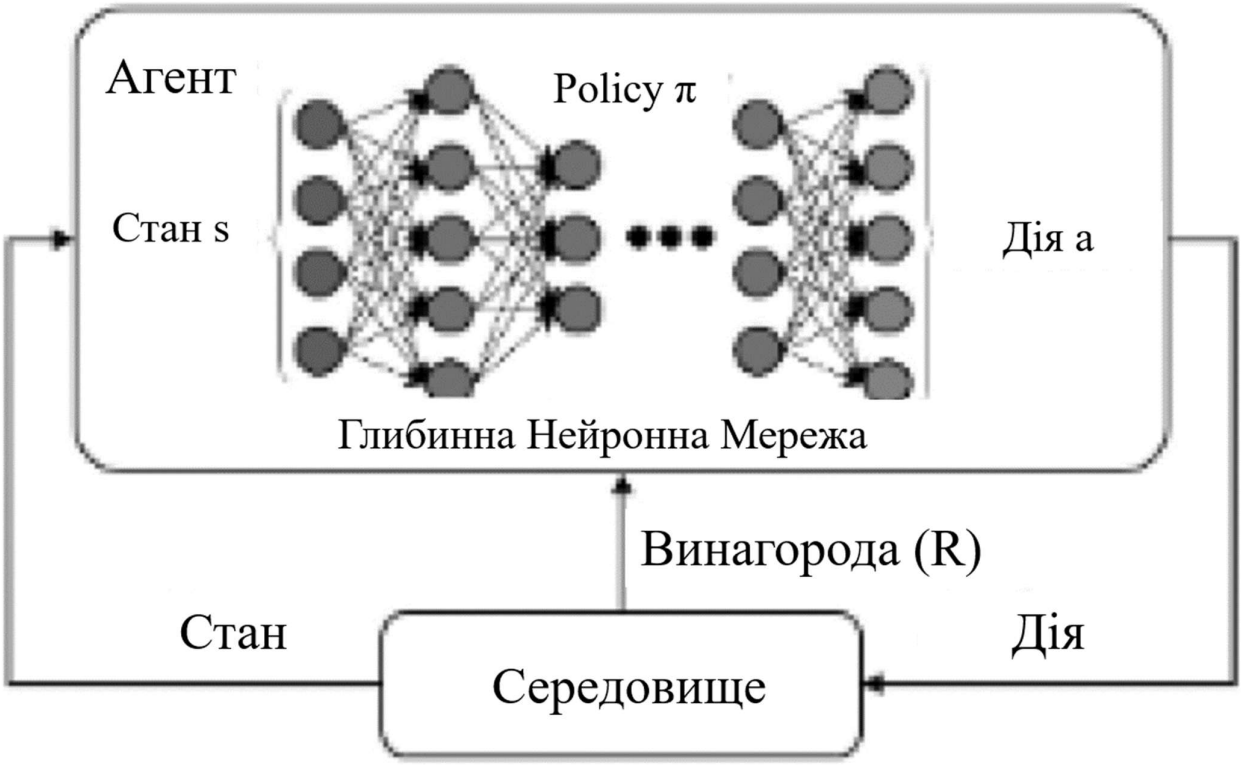

Рисунок 2.7 - Схематична структура глибинного навчання з підкріпленням, що виділяє глибинну нейронну мережу

У сфері навчання з підкріпленням на основі моделі DRL базується на вивченні моделі переходу, яка дає змогу моделювати середовище без безпосередньої взаємодії з ним, тоді як безмодельні DRL методи навчаються безпосередньо на основі взаємодії з середовищем. Q-навчання є популярною безмодельною технікою DRL для визначення найкращої політики вибору дій для будь-якого (кінцевого) марковського процесу прийняття рішень (MDP). MDP є математичною основою для моделювання рішень на основі стану, дії та винагороди. Крім того, в області використовуються Deep Q-Networks, Double DQN, Bi-directional Learning, Monte Carlo Control тощо. У методах DRL він включає моделі ГН, наприклад, глибинні нейронні мережі, засновані на принципі MDP, як

апроксиматори політики та/або функції цінності. Наприклад, CNN можна використовувати як компонент агентів BRL для навчання безпосередньо з необроблених, багатовимірних візуальних вхідних даних. У реальному світі рішення на основі DRL можна використовувати в кількох сферах застосування, включаючи робототехніку, відеоігри, обробку природної мови, комп'ютерне бачення та інших відповідних.

Протягом останніх кількох років глибинненавчання було успішно застосовано для багатьох проблем у багатьох сферах застосування. До них належать обробка природної мови, аналіз настроїв, кібербезпека, бізнес, віртуальні помічники, візуальне розпізнавання, охорона здоров'я, робототехніка та багато іншого. На рис. 2.8 узагальнено кілька потенційних реальних сфер застосування глибинного навчання.

Рисунок 2.8 - Кілька потенційних реальних сфер застосування глибинного навчання

У цих областях застосування використовуються різні методи глибинного навчання відповідно до представленої схеми на рис. 2.1, яка включає дискримінаційне навчання, генеративне навчання, а також гібридні моделі, які обговорювалися раніше. Загалом, з рис. 2.8 можна зробити висновок, що майбутні перспективи моделювання глибинного навчання в сферах реального застосування величезні, і є багато можливостей для роботи.

3 ДОСЛІДЖЕННЯ ТЕХНОЛОГІЇ ГЛИБИННОГО НАВЧАННЯ ДЛЯ ОБРОБКИ ПРИРОДНОЇ МОВИ

Область обробки природної мови (ОПМ), також відома як комп'ютерна лінгвістика, включає розробку обчислювальних моделей і процесів для вирішення практичних проблем у розумінні людських мов. Ці розчини звикли створювати корисне програмне забезпечення. Роботу в ОПМ можна розділити на дві широкі підсфери: основні області та застосування, хоча це так іноді важко чітко розрізнити, до яких областей проблема належить. Основні напрямки розглядають фундаментальні проблеми, такі як моделювання мови, яке підкреслює кількісні асоціації між природними словами; морфологічна обробка, або робота з сегментацією значущих компонентів слів і визначення справжніх частин мови слів, як вони вживаються; синтаксична обробка або розбір, який будує діаграми речень як можливі попередники семантичної обробки; і сама семантична обробка, яка намагається виділити значення слів, фраз і компонентів вищого рівня у фрагменті тексту. Сфери застосування охоплюють такі теми, як витяг корисної інформації (наприклад, іменовані сутності та зв'язки) з документів, переклад тексту між мовами та між ними, узагальнення письмових робіт, автоматичні відповіді на запитання шляхом виведення найбільш імовірних відповідей, класифікація та кластеризація документів у корпусах, а також субтитри до зображень і відео. Часто потрібно успішно впоратися з однією або декількома основними проблемами та застосувати ці ідеї та процедури для вирішення практичних проблем.

Ранні спроби ОПМ зазвичай базувалися на правилах, де правила створювалися вручну з використанням знань, отриманих з різних областей. Іноді правила були спеціальними, оскільки їх створювали для швидкого вирішення конкретних проблем. Було розроблено низку формалізмів для опису синтаксису природних мов з метою полегшення аналізу та семантичної обробки. Синтаксичні правила зазвичай поєднували з логічними семантичними правилами, щоб отримати

семантичні представлення речень, які потім використовували для вирішення практичних завдань.

Обробка на основі правил є крихкою, тому що коли люди пишуть або говорять, вони часто не дотримуються всіх тонкощів, передбачених для «правильного» використання мови. Крім того, із поширенням соціальних медіа, текстових повідомлень на телефонах і демократизацією письма в Інтернеті виникла необхідність обробляти та розуміти неформальну мову, яка свідомо порушує загальноприйняті правила орфографії та граматики. Щоб задовольнити потребу працювати з мовою понад те, що можливо за допомогою правил, написаних апріорі, ОПМ почав повільно трансформуватися, у поле, кероване даними, переважно з використанням статистичних і ймовірнісних обчислень разом з алгоритмами машинного навчання.

Основні проблеми ОПМ – це ті проблеми, які невід'ємно присутні в будь-якій комп'ютерній лінгвістичній системі. Для виконання перекладу, узагальнення тексту, створення підписів до зображень або будь-якого іншого лінгвістичного завдання необхідно певне розуміння основної мови. Це розуміння можна розбити на чотири основні області: моделювання мови, морфологія, синтаксичний аналіз і семантика.

Мовне моделювання можна розглядати двояко. По-перше, це завдання визначити, які слова слідують за якими. Однак у ширшому плані це можна розглядати як визначення значення слів, оскільки окремі слова мають лише слабке значення, якщо не безглузді, одержуючи свою повну цінність лише від їх взаємодії з іншими словами. Морфологія - це наука про те, як утворюються самі слова. Він розглядає корені слів і те, як вони розвиваються через використання префіксів і суфіксів, сполук та інших внутрішньослівних прийомів, щоб відобразити час, рід, множину та низку інших мовних конструкцій. Розбір враховує взаємодію між словами. Зокрема, він розглядає, які слова змінюють одне одного і в який спосіб, утворюючи складові (тобто фрази різних видів), що призводить до структури речення. Усе це охоплює область семантики, яка вивчає значення слів як сукупності. Він повинен враховувати значення окремих слів і те, як вони пов'язані

з іншими та змінюють їх, а також контекст, у якому ці слова з'являються, і певний ступінь світових знань, тобто «здоровий глузд».

Легко помітити, що між кожною з обговорюваних сфер є значний перетин. Тому багато проаналізованих моделей можна класифікувати як такі, що належать до кількох розділів. Як такі, вони обговорюються в найбільш релевантних розділах з логічними зв'язками з тими іншими місцями, де вони також взаємодіють.

3.1 Моделювання мови

Можливо, найважливішим завданням у поточній обробці природної мови є моделювання мови. Моделювання мови (LM) є важливою частиною майже будь-якого застосування ОПМ, від розпізнавання мовлення до машинного перекладу. По суті, мовне моделювання — це процес створення моделі для передбачення слів або простих лінгвістичних компонентів на основі попередніх слів або компонентів, зазвичай із пов'язаними ймовірностями. Це корисно для розпізнавання мовлення, оскільки може допомогти усунути неоднозначність між двома словами чи фразами, які схожі за звучанням. Подібним чином це корисно для програм, у яких користувач вводить введення, зокрема для клавіатур із сенсорним екраном, щоб забезпечити можливість прогнозування для швидкого введення тексту. Проте його потужність і універсальність випливають із того факту, що він може неявно фіксувати синтаксичні та семантичні зв'язки між словами чи компонентами в лінійному сусідстві, що робить його корисним для різноманітних завдань, таких як машинний переклад або резюмування тексту, у яких програма повинна перевіряти введені дані та /або створити вихід у формі природної мови. Використовуючи передбачення, такі програми можуть генерувати більш чіткі та людські речення та підтверджувати протилежне.

3.1.1 Раннє моделювання мови

Історично моделювання мови було статистичним завданням. Наприклад, ймовірності появи слів були розраховані з використанням даних великих корпусів. Спочатку використовувався підхід мішка слів, у якому слова просто передбачалися за частотою їх появи в навчальному корпусі, можливо, зі згладжуванням підрахунків. Пізніша розробка, модель n-грам, намагалася вирішити проблему

шляхом визначення частоти наступних слів. Це неминуче збільшило розмір моделі та обчислень, необхідних для її побудови, оскільки замість того, щоб просто мати вектори розміру v, де v було розміром словника, моделі складалися з матриць розміром v × v. Однак, оскільки багато слів рідко, якщо взагалі колись, слідують одне за одним у природному вживанні, матриці були рідкісними. Це спостереження дозволило значно зменшити розміри моделей.

Підхід n-грам розглядає не просто ймовірності слів, що йдуть за окремими іншими словами, а ймовірність слів, що йдуть після набору з n−1 попередніх слів. Отже, модель сумки слів вважатиметься уніграмою або 1-грамовою моделлю, модель, яка використовує лише одне попереднє слово, 2-грамовою або біграмною моделлю, а модель, що використовує два попередні слова, триграмною моделлю. Через зменшення ймовірності того, що набір слів одночасно зустрічається в навчальному корпусі зі збільшенням розміру n, було розроблено низку методів, які дозволяють використовувати достатньо довгі n-грами. Типові моделі використовують уніграми до 5 грамів.

3.1.2 Нейронне моделювання мови

Основною проблемою зі статистичними мовними моделями була їх нездатність добре працювати з синонімами або словами, що не входять до словникового запасу (OOV). Хоча вони могли передбачити повторну появу наборів слів, які були присутні в навчальному корпусі, вони мали труднощі з прогнозуванням нових наборів слів; наприклад, набір слів з навчального корпусу, в якому одне слово було замінено синонімом або подібним словом з іншим значенням (наприклад, автомобіль і вантажівка або червоний і синій). Прогрес було досягнуто у вирішенні цих проблем із впровадженням моделі нейронної мови. Хоча більшій частині ОПМ знадобилося ще одне десятиліття, щоб почати активно використовувати ANN, спільнота ГН негайно скористалася ними та продовжила розробку складних моделей. Вони не тільки дозволяють передбачати синонімічні слова, але також дозволяють моделювати зв’язки між словами.

Вектори слів із числовими компонентами, отримані методами мовного моделювання, називаються вкладеннями. Як правило, вбудовування слів має від 50

до 300 розмірів. Прикладом, який часто використовується, є розподілене представлення слів

3.1.3 Оцінка мовних моделей

Хоча нейронні мережі, безумовно, зробили прорив у галузі LM, часто досить важко кількісно оцінити покращення. У деяких програмах, таких як розпізнавання мовлення або машинний переклад, можна використовувати такі показники, як частота помилок у словах (WER). Коефіцієнт помилок у словах — це частка фактичних виведених слів, які не відповідають належним чином бажаним виведеним словам. Однак зазвичай бажано оцінювати мовні моделі незалежно від додатків, у яких вони з’являються. Було запропоновано ряд показників, але ідеального рішення ще не знайдено. Найбільш часто використовуваним показником є здивування, яке є оберненою ймовірністю тестового набору, нормалізованої за кількістю слів.

Здивування є розумним показником для ГН, які навчаються на тих самих наборах даних, але коли вони навчаються на різних словниках, метрика стає менш значущою. На щастя, є кілька еталонних наборів даних, які широко використовуються в цій галузі, що дозволяє проводити розумне порівняння. Два таких набори даних є Penn Treebank (PTB), Billion Word Benchmark.

3.1.4 Мережі пам'яті та механізми уваги в моделюванні мови

Перша навчена мережа мала простий механізм уваги, який не був повністю підключений, маючи вікно довжиною п’ять. Друга і третя мережі були більш новими, особливо в контексті мовного моделювання. Дослідники припустили, що використання єдиного значення для прогнозування наступного токена, кодування інформації для блоку концентрації уваги та декодування інформації в блоку концентрації уваги заважає мережі, оскільки це різні завдання, і важко навчити один параметр виконувати всі три завдання одночасно. Таким чином, у другій мережі вони розробили кожен вузол, щоб мати два виходи: один використовується для кодування та декодування інформації в блоку уваги, а інший для явного прогнозування наступних токенів. У третій мережі вони додатково розділили виходи, використовуючи окремі значення для кодування інформації, що надходить

в блок уваги, і декодування інформації, яка з нього витягується. Ці мережі були відповідно названі мережами «Ключ-Значення Увага» і «Ключ-Значення-Передказування уваги».

Механізм уваги покращив здивування порівняно з базовим рівнем, і що послідовне додавання другого та третього параметрів призвело до подальшого збільшення. Це показало, що використання механізмів уваги, особливо з кількома різними параметрами, є корисним у мовному моделюванні. Було також зазначено, що лише попередні п’ять чи близько того токенів мали значну цінність (тому вибрано п’ять розмірів вікна). Тому було вирішено протестувати четверту мережу, яка просто використовувала залишкові з’єднання від кожного з п’яти попередніх блоків. Було виявлено, що ця мережа також забезпечила результати, які можна порівняти з багатьма більшими RNN та LSTM, що свідчить про те, що розумні результати можуть бути досягнуті за допомогою простіших мереж, ніж це часто пропонувалося.

Ще одне нещодавнє дослідження було проведено щодо використання мереж залишкової пам’яті (RMN) для мовного моделювання. Хоча автори нічого не згадали про те, чи тестували вони використання кількох залишкових з’єднань, вони виявили, що залишкові з’єднання, які пропускають два шари, були найефективнішими, за ними йдуть ті, що пропускають один шар. Використовуючи ці висновки, вони розробили рекурентні мережі із залишковими з’єднаннями, пропускаючи дві одиниці одночасно. Замість того, щоб такий зв’язок був присутній на кожному рівні, у них такі повторення повторювалися блоками. Залишковий зв'язок був присутній між першим шаром і четвертим, як і між п'ятим шаром і восьмим, а також між дев'ятим і дванадцятим. Було виявлено, що збільшення глибини мережі покращило результати, але при використанні великих розмірів пакетів спостерігалися обмеження пам’яті. Виявилося, що ширина мережі не має особливого значення з точки зору продуктивності, однак виявилося, що широкі мережі важче навчити. Мережі було навчено на Penn Treebank і порівняно з результатами RNN і LSTM. Було виявлено, що RMN здатні перевершувати LSTM аналогічного розміру.

3.1.5 Згорткові нейронні мережі в моделюванні мови

Згорточна нейронна мережа, яка нещодавно використовувалася в мовному моделюванні, була такою, в якій шари об'єднання були замінені на повністю зв'язані рівні. Ці рівні дозволили скоротити карти функцій до просторів нижчих розмірів, як це зазвичай роблять шари об'єднання. Однак у той час як будь-які посилання на розташування таких функцій втрачаються на рівнях об'єднання, повністю з'єднані рівні певною мірою зберігають цю інформацію. Було випробувано кілька варіантів CNN, усі з яких включали цю характеристику. Загалом було реалізовано три різні архітектури: багаторівневий персептрон CNN (MLPConv), у якому фільтри були не просто лінійними, а малими MLP; багаторівневий CNN (ML-CNN), в якому кілька згорткових рівнів накладаються один на одного; і комбінація цих мереж під назвою COM, у якій розміри ядра для фільтрів змінювалися (у цьому випадку вони становили три та п'ять). Усі мережі були протестовані на трьох різних наборах даних: Penn Treebank, Europarl-NC. Результати цього дослідження показали, що накопичення згорткових шарів насправді завдало шкоди моделюванню мови, але MLPConv і COM були ефективними для зменшення здивування на кожному з тестованих наборів даних. Крім того, поєднання MLPConv із різними розмірами ядра COM забезпечило ще кращі результати, досягнувши дев'яносто одного здивування в Penn Treebank. Дослідники провели аналіз, який показав, що мережі засвоїли певні шаблони слів, наприклад «як . . . як». Нарешті, це дослідження показало, що CNN можна використовувати для фіксації довготривалих залежностей у реченнях. Виявилося, що найбільше значення мають ближчі слова, але слова, розташовані далі, також мають певне значення. Враховувалися слова на відстані до шістнадцяти позицій.

3.1.6 Моделі нейронної мови з урахуванням символів

У той час як більшість CNN, що використовуються в ОПМ, отримують слова (або вбудовування слів) як вхідні дані, деякі останні мережі замість цього аналізують введення на рівні символів. Однією з таких мереж є мережа, яка, на відміну від попередніх мереж, які приймали введення на рівні символів, приймали лише такі введення, а не поєднували його з вбудованими словами. CNN

використовувався для обробки введення на рівні символів, щоб забезпечити представлення слів. Подібним чином, як це зазвичай буває при вбудовуванні слів, ці представлення потім подавали в пару кодер-декодер, що складається з мережі магістралей (мережі із закритими шлюзами, що нагадує LSTM) і LSTM. Вони навчили мережу на англійському Penn Treebank, а також на наборах даних для чеської, німецької, іспанської, французької, російської та арабської мов. Дані арабською мовою були взяті з корпусу News-Commentary, а всі інші дані – з семінару ACL з машинного перекладу 2013 року (як малі, так і великі набори даних). Для всіх неанглійських мов, окрім російської, мережа перевершила раніше опубліковані результати як у великих, так і в малих наборах даних. Для російської мови кращі результати були досягнуті на невеликому наборі даних, але не на великому. На Penn Treebank результати були отримані на рівні з існуючим сучасним рівнем техніки, досягнувши здивування 78,9 (порівняно з 78,4). Проте мережа мала лише 19 мільйонів параметрів, які можна навчити, що значно менше, ніж інші, і була обмежена кількістю слів, позначених як невідомі в Penn Treebank. Оскільки мережа була зосереджена на морфологічних подібностях, отриманих за допомогою аналізу рівня символів, вона була більш здатною, ніж попередні моделі, обробляти слова, які рідко можна побачити в корпусі. Аналіз показав, що без використання шарів магістралі багато слів мали найближчих сусідів, орфографічно схожих (написаних подібно), але не обов'язково семантично подібних. Додавання рівнів магістралі (які використовувалися в найефективнішій мережі), здавалося, вирішило цю проблему. Крім того, мережа була здатна розпізнавати слова з орфографічними помилками або слова, написані нестандартним чином (наприклад, looooook замість look), а також розпізнавати слова зі словникового запасу. Аналіз також показав, що мережа здатна ідентифікувати префікси, корені та суфікси, а також розуміти слова з дефісами, що робить її надзвичайно надійною моделлю.

3.2 Морфологія

Морфологія займається пошуком сегментів в окремих словах, включаючи корені та основи, префікси, суфікси та, у деяких мовах, інфікси. Афікси (префікси, суфікси чи інфікси) використовуються для явної зміни основ роду, числа, особи

тощо. Крім того, різні основи, що походять від одного кореня, зазвичай передають окремі, але пов’язані поняття.

Luong та ін. побудували мовну модель, яка була морфологічно усвідомленою. RvNN використовувався для моделювання морфологічної структури англійських слів. Потім модель нейронної мови була розміщена поверх RvNN. Модель була навчена на наборі даних WordSim-353, а сегментація була виконана за допомогою Morfessor. Було створено додатковий набір даних рідкісних слів, оскільки більшість доступних наборів даних погано моделюють ці слова. Було створено дві моделі — одна з використанням контексту, а інша — ні. Було виявлено, що модель, яка була нечутлива до контексту, надмірно враховувала певні морфологічні структури. Зокрема, слова з однаковою корою були згруповані разом, навіть якщо вони були антонімами. Контекстно-залежна модель працювала краще, відзначаючи зв’язки між основами, але також враховуючи інші особливості, такі як префікс «un». Модель також була протестована на кількох інших популярних наборах даних, значно перевершуючи попередні моделі вбудовування в усіх.

Хороший морфологічний аналізатор часто важливий у контексті більших лінгвістичних систем та інших завдань ОПМ. Таким чином, одне нещодавнє дослідження Бєлінкова та ін. досліджували ступінь вивчення та використання морфології різноманітними моделями нейронного машинного перекладу. У цьому дослідженні було створено низку моделей перекладу, усі з яких перекладаються з англійської на французьку, німецьку, чеську, арабську чи іврит. Кодери та декодери були моделями на основі LSTM (деякі з механізмами уваги) або символьними CNN, і моделі були навчені на WIT. Потім декодери були замінені тегерами частини мови (POS) і морфологічними тегерами, фіксуючи ваги кодерів, щоб зберегти внутрішні представлення. Впливи кодерів досліджувалися, як і ефекти декодерів, підключених під час навчання. Дослідження прийшло до висновку, що використання механізмів уваги знижує продуктивність кодерів, але підвищує продуктивність декодерів. Крім того, було виявлено, що моделі з розпізнаванням символів є кращими за інші для вивчення морфології та що вихідна мова впливає

на продуктивність кодерів. Зокрема, чим більш морфологічно багата вихідна мова, тим гірші представлення, створені кодерами.

Моріта та ін. проаналізували нову морфологічну модель мови для несегментованих мов, таких як японська. Вони побудували модель на основі RNN з декодером пошуку променів і навчили її як на корпусі з автоматичною міткою, так і на новому корпусі з мітками вручну, який вони представили. Модель спільно виконувала низку завдань, включаючи морфологічний аналіз, тегування POS та лемматизацію. Потім модель була протестована на Kyoto Text Corpus, перевершивши всі базові показники для всіх завдань.

3.3 Парсинг

Парсинг перевіряє, як різні слова та фрази співвідносяться між собою в реченні. Існує принаймні дві різні форми синтаксичного аналізу: синтаксичний аналіз конституентів і аналіз залежностей. Під час синтаксичного аналізу складових фраз фразові складові виділяються з речення в ієрархічному порядку. Визначаються фрази, які, у свою чергу, утворюють більші фрази, які зрештою завершуються повними реченнями. З іншого боку, аналіз залежностей дивиться виключно на зв’язки між парами окремих слів.

Найновіші випадки використання глибинного навчання в синтаксичному аналізі природної мови були в області синтаксичного аналізу залежностей, усередині якого існує ще один великий розрив у типах рішень. Синтаксичний аналіз на основі графів створює кілька дерев синтаксичного аналізу, які потім шукаються, щоб знайти правильне. Більшість підходів на основі графів є генеративними моделями, у яких формальна граматика, заснована на природній мові, використовується для побудови дерев. Без такого підходу виникла б величезна кількість дерев, багато з яких були б абсолютно нелогічними.

Більш популярними в останні роки, ніж підходи на основі графіків, були підходи на основі переходів. У цих підходах зазвичай будується лише одне дерево аналізу. Хоча було запропоновано ряд модифікацій, стандартний метод синтаксичного аналізу залежностей на основі переходів полягає у створенні буфера, що містить усі слова в реченні, і стека, що містить лише мітку ROOT. Потім

слова надходять у стек, де між двома верхніми елементами створюються з'єднання, відомі як дуги. Ці дуги можуть бути як правими, так і лівими, залежно від того, чи залежить верхнє слово (яке далі праворуч у реченні) від нижнього слова (яке далі ліворуч), чи нижнє слово залежить від верхнього . Після визначення залежностей слова витягуються зі стеку. Процес триває, доки буфер не спорожніє, а в стеку залишиться лише мітка ROOT. Для регулювання умов, у яких відбувається кожна з описаних раніше дій, використовуються три основні підходи. У дуговому стандартному підході усі залежні зв'язуються зі словом до того, як слово з'єднується зі своїм батьківським словом. Згідно з підходом, що рветься до дуги, слова пов'язуються зі своїми батьками якнайшвидше, незалежно від того, чи всі їхні діти пов'язані з ними. Нарешті, у підході Swap-Lazy стандартний підхід дуги змінено, щоб дозволити міняти місцями в стеку. Це робить можливим побудову графіків непроективних ребер.

3.3.1 Ранній нейронний парсинг

Порівняно з багатьма сферами ОПМ, синтаксичний аналіз лише нещодавно був введений у силу глибинного навчання. Один із ранніх застосувань глибинного навчання для обробки природної мови, включав використання RNN з ймовірнісними контекстно-вільними граматиками (PCFG), що дозволяло розбирати семантично різні фрази по-різному, навіть якщо їхні теги частини мови ідентичні. Наскільки відомо авторам, першою нейронною моделлю, яка досягла найсучаснішого аналізу, була модель Le та Zuidema. Така ефективність була досягнута в Penn Treebank як для міченої оцінки прикріплення (LAS), так і для оцінки неміченої прикріпленості (UAS) за допомогою рекурсивної нейронної мережі Inside-Out, яка використовувала два векторних представлення (внутрішнє та зовнішнє), щоб дозволити обидва верхні потоки даних вниз і знизу вгору. Віньялс та ін. створили LSTM з механізмом уваги в синтаксичному синтаксичному аналізаторі конституентів, який вони протестували на даних з доменів, відмінних від доменів тестових даних на відміну від частини Penn Treebank у Wall Street Journal, показуючи, що нейронні моделі можуть узагальнювати між областями. Стенеторп вперше використав вбудовування для розбору залежностей, проклавши

шлях для останніх спроб розбору природної мови. Цей підхід використовував RNN для створення спрямованого ациклічного графа. Незважаючи на те, що ця модель давала результати в межах 2% від сучасного рівня, виданому Wall Street Journal, до моменту, коли вона досягла кінця речення, здавалося, йому було важко запам’ятати фрази з початку речення.

3.3.2 Парсинг залежностей на основі переходів

Основою для найбільших недавніх досліджень у галузі синтаксичного аналізу є робота Чена та Меннінга, які розповсюдили сучасний рівень UAS і LAS на англійських і китайських наборах даних, набравши 92,2% UAS на англійському Penn Treebank. Вони досягли цього, використовуючи просту нейронну мережу прямого зв’язку як засіб прийняття рішень у аналізаторі на основі переходів. Таким чином вони змогли підірвати проблему розрідженості статистичних моделей, які активно використовувалися. Хоча вони досягли найсучасніших результатів, вони відзначили простоту своєї моделі, а також низку областей, у яких її можна вдосконалити.

Першою з таких областей був пошуковий механізм, використовуваний моделлю. Чен і Меннінг використовували простий жадібний пошук з променевим пошуком. Зробивши це, вони досягли значного покращення, наблизившись до продуктивності нового рівня техніки. На додаток до використання пошуку за променем, Weiss et al покращив роботу Чена та Меннінга, використовуючи глибшу нейронну мережу із залишковими зв’язками та рівень персептрона, розміщений після рівню softmax. Крім того, вони змогли навчитися на значно більшій кількості прикладів, ніж зазвичай, за допомогою три-навчання, процесу, під час якого потенційні зразки даних передаються двом іншим синтаксичним аналізаторам, і ті, щодо яких обидва парсери погоджуються. Цей підхід став найсучаснішим із відповідними оцінками UAS і LAS 94,26% і 92,41% на Penn Treebank. Альберті та ін. далі розширили цю роботу, протестувавши кілька подібних моделей на наборі даних Wall Street Journal, наборі даних Web English Treebank і даних Question Treebank, а також на наборах даних з CoNLL 2009 для ряду мови. Найсучасніша

продуктивність була досягнута як в UAS, так і в LAS на трьох наборах даних англійською мовою, а також для більшості мов, перевірених у CoNLL 2009.

Інша модель була створена з використанням LSTM замість прямої мережі. На відміну від попередніх моделей, ця модель мала знання про весь буфер і весь стек, а також про всю історію рішень про перехід. Це дозволило зробити кращі прогнози, генеруючи найсучасніші UAS і LAS оцінки 93,1% і 90,9% відповідно в Stanford Dependency Treebank, а також найсучасніші результати китайського набору даних CTB5. Нарешті, Andor et al. використовував мережу прямого зв'язку з глобальною нормалізацією для ряду завдань, включаючи тегування частин мови, стиснення речень і аналіз залежностей. За всіма завданнями було отримано найсучасніші результати, у тому числі досягнуто результату UAS 94,61% за набором даних Wall Street Journal. Примітно, що для досягнення цих результатів їх модель вимагала значно менше обчислень, ніж аналогічні моделі.

3.3.3 Пасинг генеративних залежностей та складових

Хоча розбір залежностей на основі переходів все ще залишається успішним, наразі визначаючи сучасний рівень, нещодавно були досягнуті покращення шляхом застосування глибинного навчання до аналізу складових і генеративних моделей. Dyer та ін. запропонував модель, яка використовує рекурентні граматики нейронної мережі для синтаксичного аналізу та моделювання мови. У той час як більшість підходів використовують висхідний підхід до синтаксичного аналізу, цей використовує підхід зверху вниз, приймаючи як вхідні дані повне речення на додаток до поточного дерева синтаксичного аналізу. Це дозволило розглядати речення як ціле, а не просто дозволяти розглядати місцеві фрази в ньому. Незважаючи на те, що ця модель не досягла найсучасніших результатів у всіх розборах, ця модель досягла найкращих результатів у генеративному розборі англійської мови, а також у моделюванні мови одного речення. Він також досяг результатів, близьких до найкращих у китайському генеративному аналізі.

Першою моделлю, яка досягла найсучасніших результатів за допомогою підходу глибокого навчання до генеративного синтаксичного аналізу, була модель Чое та Чарняка. Розбір розглядався як проблема моделювання мови, і LSTM

використовувався для призначення ймовірностей деревам розбору. Використовуючи цей підхід, на Penn Treebank було досягнуто балів UAS і LAS 95,9% і 94,1% відповідно. Фрід та ін. розглянули обидві ці моделі та протестували їх, щоб визначити, чи походить потужність цих моделей від процесу реранжування чи просто від об'єднаної потужності двох моделей. Вони виявили, що хоча використання одного синтаксичного аналізатора для створення дерев-кандидатів і іншого для їх ранжування було кращим за підхід з одним парсером, явне поєднання двох парсерів все ж було кращим. Вони використали два синтаксичних аналізатора, щоб відібрати кандидатів і змінити їх рейтинг, досягнувши найсучасніших результатів. Вони розширили цю модель для використання трьох синтаксичних аналізаторів, досягнувши навіть кращих результатів, з показником F1 на Penn Treebank 93,4%. Нарешті, ансамбль із восьми таких моделей (з використанням двох синтаксичних аналізаторів) був створений і досяг найкращих результатів (94,25% F1 на Penn Treebank) перед моделлю, запропонованою Вангом та ін.

3.4 Семантика

Семантична обробка передбачає розуміння значення слів, фраз, речень або документів на певному рівні. Термін «значення» важко визначити, і лінгвісти та філософи сперечаються про це століттями. Вбудовані слова, такі як Word2Vec і GloVe, стверджують, що фіксують значення слів відповідно до гіпотези розподілу значення. Як наслідок, коли вектори, що відповідають фразам, реченням або іншим компонентам тексту, обробляються за допомогою нейронної мережі, представлення, яке можна умовно вважати семантично репрезентативним, обчислюється композиційно. Така композиція необхідна для виконання багатьох завдань, таких як підбиття підсумків, відповіді на питання та відеозаписи. У цьому розділі дослідження нейронної семантичної обробки розділено на дві окремі області: робота, зосереджена на порівнянні семантичної подібності двох частин тексту, і робота, зосереджена на захопленні та передачі значення в складових мови високого рівня, зокрема в реченнях.

3.4.1 Семантичне порівняння

Один із способів перевірити ефективність підходу до обчислення семантики — побачити, чи дві подібні фрази, речення чи документи, які люди вважають подібними, також оцінюються програмою однаково.

Ху та ін. запропонував два CNN для виконання завдання семантичного порівняння. Перша модель, ARC-I, використовувала «сіамську» мережу, у якій два CNN, що поділяли ваги, оцінювали два речення паралельно. У другій мережі з'єднання було встановлено між двома, дозволяючи обмінюватися до остаточного стану CNN. Було проведено три експерименти з «множинним вибором»: тестування завершення речення англійською мовою, відповіді на китайські твіти та порівняння значення речення англійською мовою. Цей підхід перевершив ряд існуючих моделей.

Також використовували CNN для порівняння семантичної подібності речень. Створені карти функцій потім порівнювали за допомогою «рівню вимірювання подібності», за яким слідував шар повного зв'язку, а потім вихідний рівень log-softmax. Довжина вікон, використовуваних у згорткових рівнях, коливалася від одного до чотирьох. Мережа була навчена та оцінена на трьох різних наборах даних: MSRP, набір даних Sentences Involving Compositional Knowledge (SICK) і Microsoft Video Paraphrase Corpus (MSRVID). На першому та третьому були досягнуті найсучасніші результати, а на другому – майже рівні. Дослідження абляції показали, що низка елементів, таких як теги POS, численні операції об'єднання та численні показники подібності, а також кілька інших функцій, усі вони сприяли продуктивності моделі.

3.4.2 Моделювання речень

Хоча порівняння значень речень і фраз є корисним, воно значною мірою залежить від можливості обчислити значення окремих речень і фраз. Це загальна тема для всіх ОПМ. Розширюючись від моделювання нейронної мови, яке намагається вловити значення слів у векторах, моделювання речень намагається вловити значення речень у векторах. Зробивши крок далі, є такі моделі, які намагаються таким чином моделювати абзаци або більші частини тексту.

Незважаючи на те, що вони інтригують, ще багато роботи потрібно зробити на рівнях фраз і речень.

Безсумнівно, сфера простору ОПМ, яка найбільше залежить від хорошого семантичного розуміння, це машинний переклад. Системи нейронного машинного перекладу (NMT) є чудовим випробувальним майданчиком для дослідження внутрішніх семантичних уявлень. Кодерів було навчено працювати з чотирма різними мовними парами: англійською та арабською, англійською та іспанською, англійською та китайською, а також англійською та німецькою. Спираючись на низку попередніх досліджень, класифікатори декодування були навчені на чотирьох різних наборах даних: Multi-NLI, який є розширеною версією SNLI, а також три набори даних від JHU Decompositional Semantics Initiative. Жоден із результатів не був особливо сильним, хоча вони були найсильнішими в SPR. Це привело дослідників до висновку, що моделі NMT погано справляються із захопленням перефразованої інформації та не в змозі зафіксувати умовиводи, які допомагають розрізняти анафору (наприклад, розрізняти стать, множину тощо). Однак вони виявили, що моделі можуть дізнатися достатню кількість проторолей (наприклад, хто або що є одержувачем дії).

Було створено кілька мереж кодера-декодера на основі LSTM, і отримані вектори вбудовування були проаналізовані. Використовувався єдиний кодер, який приймав англійські речення як вхідні дані, а також чотири різних декодери. Першим таким декодером був реплікувальний декодер, який намагався відтворити оригінальний англійський вхід. Другий і третій декодери намагалися перекласти текст німецькою та французькою мовами. Нарешті, четвертим декодером був POS тегер. Використовувалися різні комбінації декодерів; одна модель мала лише реплікувальний декодер, тоді як інші мали два, три або, в одному випадку, усі чотири. Речення чотирнадцяти різних структур із набору даних EuroParl використовувалися для навчання мереж. Набір тестових речень потім подавався на кодери та аналізувався їхній вихід. У всіх випадках було сформовано чотирнадцять кластерів, кожен з яких відповідає одній із структур речень. Аналіз відстаней між кластерами та кількості неправильно класифікованих результатів показав, що

додавання більшої кількості декодерів призвело до більш правильних і чіткіших кластерів. Зокрема, використання всіх чотирьох декодерів призвело до нульового рівня помилок. Крім того, дослідники перевірили та підтвердили гіпотезу щодо цих вбудованих речень. Вони виявили, що так само, як логічну арифметику можна виконати над вбудованими словами, так само її можна виконати і над вбудованими реченнями. Хоча визнається, що це добре контрольоване середовище, воно показує, що моделі ОПМ здатні вловлювати багато сенсу та відкривають шлях для нового напрямку досліджень і, можливо, ще однієї революції в цій галузі.

Отже, підходи до глибинного навчання загалом показали дуже хороші результати у створенні основи, на якій можуть створюватися корисні програми на природній мові. У цьому розділі представлено лише кілька вибраних спроб використання глибинного навчання для вирішення таких проблем. Теми, які не обговорюються, включають позначення частин мови, виправлення орфографії, усунення неоднозначності слів, вирішення співпосилань, дискурс та розмовні проблеми та багато інших.

3.5 Обґрунтування вибору технології для стеганографічних перетворень

Сенс застосування глибинного навчання для обробки природних мов полягає в тому, що глибинні нейронні мережі виконують роботу, на здійснення котрої впродовж прийнятного проміжку часу потрібно було б застосовувати десятки чи навіть сотні команд професійних лінгвістів. Традиційні нейронні мережі не мають можливості приймати поточні рішення на основі своїх попередніх суджень. Велика кількість задач, що вирішуються при машинній обробці природніх мов потребує поетапного аналізу даних з врахуванням попередніх результатів. Нейронна мережа повинна «читати» речення слово за словом, «осмислюючи» його значення виходячи з контексту.

Рекурентні нейронні мережі – це мережі, що містять зворотні зв’язки і дозволяють зберігати інформацію.

4 ПРОГРАМНА РЕАЛІЗАЦІЯ ПРОТОТИПУ ЛІНГВІСТИЧНОЇ СТЕГАНОСИСТЕМИ ІЗ ЗАСТОСУВАННЯМ ТЕХНОЛОГІЇ ГЛИБИННОГО НАВЧАННЯ

З метою досягнення приховування великої ємності текстової інформації, гарантуючи природність стеганографічних текстів, тобто гарантуючи ємність для приховування, забезпечуючи при цьому достатньо високий ступінь приховування, в цій роботі проводиться текстова стеганографія на основі рекурентних нейронних мереж, які можуть автоматично генерувати високоякісні тексти на основі вхідного бітового потока.

4.1 Обґрунтування вибору мови програмування та середовища розробки

Для реалізації запропонованого алгоритму було вибрано мову програмування Python, використовуючи бібліотеку TensorFlow. TensorFlow є обчислювальною бібліотекою для побудови моделей машинного навчання. Ця бібліотека забезпечує безліч різних наборів інструментів, які дозволяють писати код на бажаному рівні абстракції.

TensorFlow може навчати та запускати глибокі нейронні мережі для класифікації рукописних цифр, розпізнавання зображень, вбудовування слів, рекурентні нейроні мережи, моделі послідовность-в-послідовність для машинного перекладу, обробки природної мови та моделювання на основі диференціального рівняння.

TensorFlow дозволяє створювати графи потоків даних - структури, що описують, як дані рухаються через граф або серію вузлів обробки. TensorFlow забезпечує все це за допомогою мови Python. Мова Python проста у вивченні та роботі з нею, і вона пропонує зручні способи висловити, як абстракції високого рівня можна поєднати. Вузли та тензори в TensorFlow - це об’єкти Python, а програми TensorFlow - самі програми Python.

Найбільшою перевагою TensorFlow для розвитку машинного навчання є абстракція. Замість того, щоб займатися дрібницями, деталями реалізації

алгоритмів або з'ясовувати правильні способи приєднання виходу однієї функції до входу іншої, можна зосередитись на загальній логіці програми. TensorFlow «піклується» про деталі самостійно.

TensorFlow пропонує додаткові зручності для налагодження та отримання самоаналізу у програмах TensorFlow. Режим енергійного виконання дозволяє оцінювати та модифікувати кожну операцію з графіком окремо та прозоро, замість того, щоб будувати весь графік як єдиний непрозорий об'єкт і оцінювати його всі відразу.

TensorFlow також отримує багато переваг завдяки підтримці комерційного спорядження зі списку A у Google.

Отже, бібліотека TensorFlow є найкращими вибором бля реалізації запропонованого в роботі алгоритму, бо має всі необхідні для цього функціональні можливості.

4.2 Експериментальні дослідження алгоритмів приховування та вилучення повідомлень за розробленими стеганосистемами

Детальне пояснення запропонованої моделі показана на рисунку 4.1.

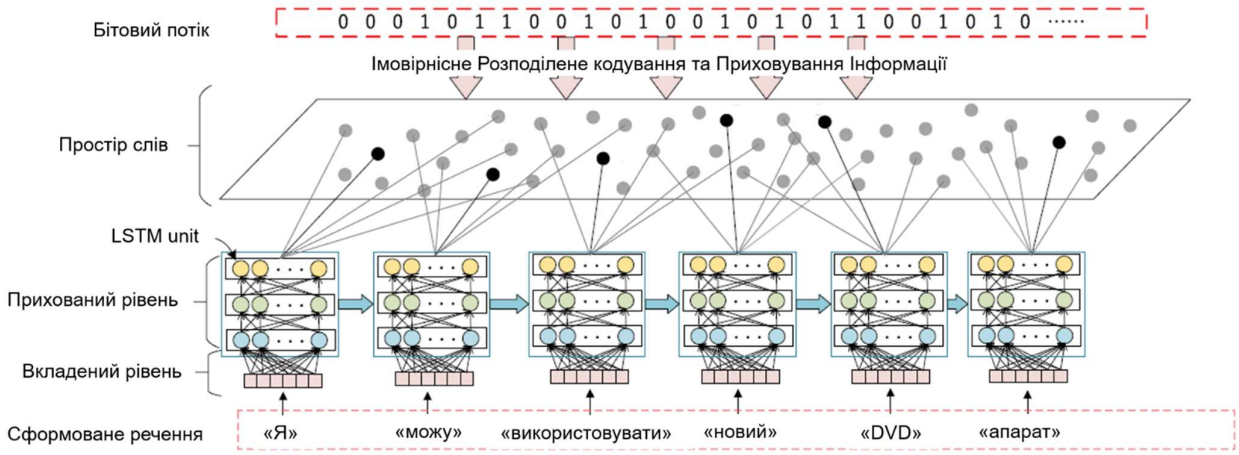

Рисунок 4.1 – Детальне пояснення запропонованої моделі та алгоритму приховування інформації.

Модель складається з трьох основних модулів: модуль автоматичного формування тексту, модуль приховування інформації та модуль вилучення інформації.

Модуль автоматичної генерації тексту призначений в основному для моделювання природного тексту та використовування здатності самонавчання нейронної мережі для вивчення доброї статистичної мовної моделі із великої кількості зразків, а також для оцінки умовної ймовірності розподілу у кожен момент.

Модуль приховування інформації усвідомлює можливість належного приховування секретного потоку бітів, кодуючи умовну ймовірність розподілу в кожен момент, включаючи кодування фіксованої довжини (англ. fixed-length encoding (FLC)) та кодування змінної довжини (англ. variable-length encoding (VLC)).

Модуль вилучення інформації імітує приймальний кінець. Після отримання природного тексту, згенерованного за запропонованою моделлю, в який вбудована секретна інформація, модулю потрібно розшифрувати текст та отримати конфіденційне повідомлення.

У верхній частині моделі запропоновано бітовий потік, який необхідно приховати, в нижній частині показано природне речення, сформоване рекурентною нейронною мережею на основі інформації, що приховується. Схема саме РНМ, яка використовується в алгоритмі, показана в середині моделі. Вона містить вбудований рівень, кілька мережевих рівнів з декількома LSTM модулями та вихідний рівень. Вихідний рівень – це ймовірнісний розподіл наступного слова. Імовірнісний розподіл слів кодується, а потім вибирається відповідне до секретного потоку бітів слово, щоб досягти мети приховування інформації.

4.2.1 Автоматична генерація тексту на основі РНМ

В процесі автоматичного формування тексту ми в основному беремо переваги потужної здатності РНМ у виділенні функції та вираз для послідовних сигналів. Це може запобігти сигналам в попередні моменти і обчислити ймовірнісний розподіл сигналу за часом. Кожне речення може розглядатися як послідовний сигнал, а кожне слово може розглядатися як сигнал на визначеному етапі часу. Коли для генерації речень використовується LSTM, ми вводимо кожне слово на відповідний етап часу. Перший рівень нейронної мережі – рівень

вбудовування, він відображає кожне слово до щільного семантичного простору з розмірністю *d*. Тому, кожне речення можна представити як матрицю, де *i*-ий рядок вказує на *i*-те слово в реченні *S*, с длиною *L*(1).

$$S = \begin{bmatrix} \text{Слово}_{S_1} \\ \text{Слово}_{S_2} \\ \cdots \\ \text{Слово}_{S_L} \end{bmatrix} = \begin{bmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,d} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,d} \\ \vdots & \vdots & \ddots & \vdots \\ x_{L,1} & x_{L,2} & \cdots & x_{L,d} \end{bmatrix} \quad (4.1)$$

Взагалі, рекурентна нейронна мережа складається з декількох мережевих рівнів, кожен з яких має декілька модулів LSTM. Більша кількість таких модулів нейронної мережі забезпечує сильнішу здатність до вилучення. Отже, ми складаємо мережу з декількома рівнями модулів LSTM.

Всі параметри нейронної мережі, включаючи кожен слово-вектор, потрібно отримати за допомогою навчання. Для того щоб отримати статистичну мовну модель, яка відповідає навчанню за зразками, також визначається функція втрати всієї мережи як негативний лог статистичної ймовірності кожного речення.

У процесі навчання ми оновлюємо параметри мережі за допомогою алгоритму зворотного розповсюдження. Після мінімізації функції втрат за допомогою ітеративної оптимізації мережі, ми отримаємо мовну модель, яка все більше відповідає статистичним характеристикам навчальних зразків.

4.2.2 Алгоритм приховування інформації

У модулі приховування інформації ми в основному кодуємо кожне слово на основі їх умовного розподілу ймовірності, щоб сформувати взаємозв'язок відображення від двійкового бітового потоку до простору слів. Ідея базується головним чином на основі того, що коли модель добре навчена, тоді існує більше ніж одне можливе рішення в кожен момент часу. Після спадання прогнозування ймовірності всіх слів у словнику, ми можемо вибрати найкращі відсортовані слова для побудови пулу кандидатів (ПК). Тобто, коли вибирається відповідний розмір пулу кандидатів, будь-яке слово в ПК, вибране у той момент як вихідне, є розумним і не вплине на якість тексту, що генерується, тому воно стає місцем, де можна скрити інформацію. Важно зазначити, що в кожну мить обираються різні слова,

тобто, ймовірнісний розподіл слів буде інший на наступному етапі часу. Після того, як ми отримаємо пул кандидатів, нам потрібно знайти ефективний метод кодування для кодування слів у ньому.

Очевидно, що різні методи кодування вплинуть на кінцевий результат всього алгоритму. У роботі пропонується два методи: кодування фіксованої довжини, в основі якого є ідеальне бінарне дерево; кодування змінної довжин, в основі якого є дерево Хаффмана. Ідеальне бінарне дерево – це двійкове дерево, в якому всі внутрішні вузли мають двох дітей і всі листя мають однакову глибину. Дерево Хаффмана також є бінарним деревом. Різниця в тому, що воно потребує більше врахування розподілу ймовірностей кожного вихідного символу у процесі будівництва, і може забезпечити, що довжина коду, необхідна для символу з вищою ймовірністю кодування, є коротшою [15]. І в, і в кожне слово в пулі кандидатів представляється з кожним вузлом листа дерева, краї з’єднують кожний нелистовий вузол (включаючи кореневий вузол), а його два дочірні вузли потім кодуються 0 і 1, відповідно, 0 – ліворуч, 1 – праворуч, як показано на рисунку 4.2.

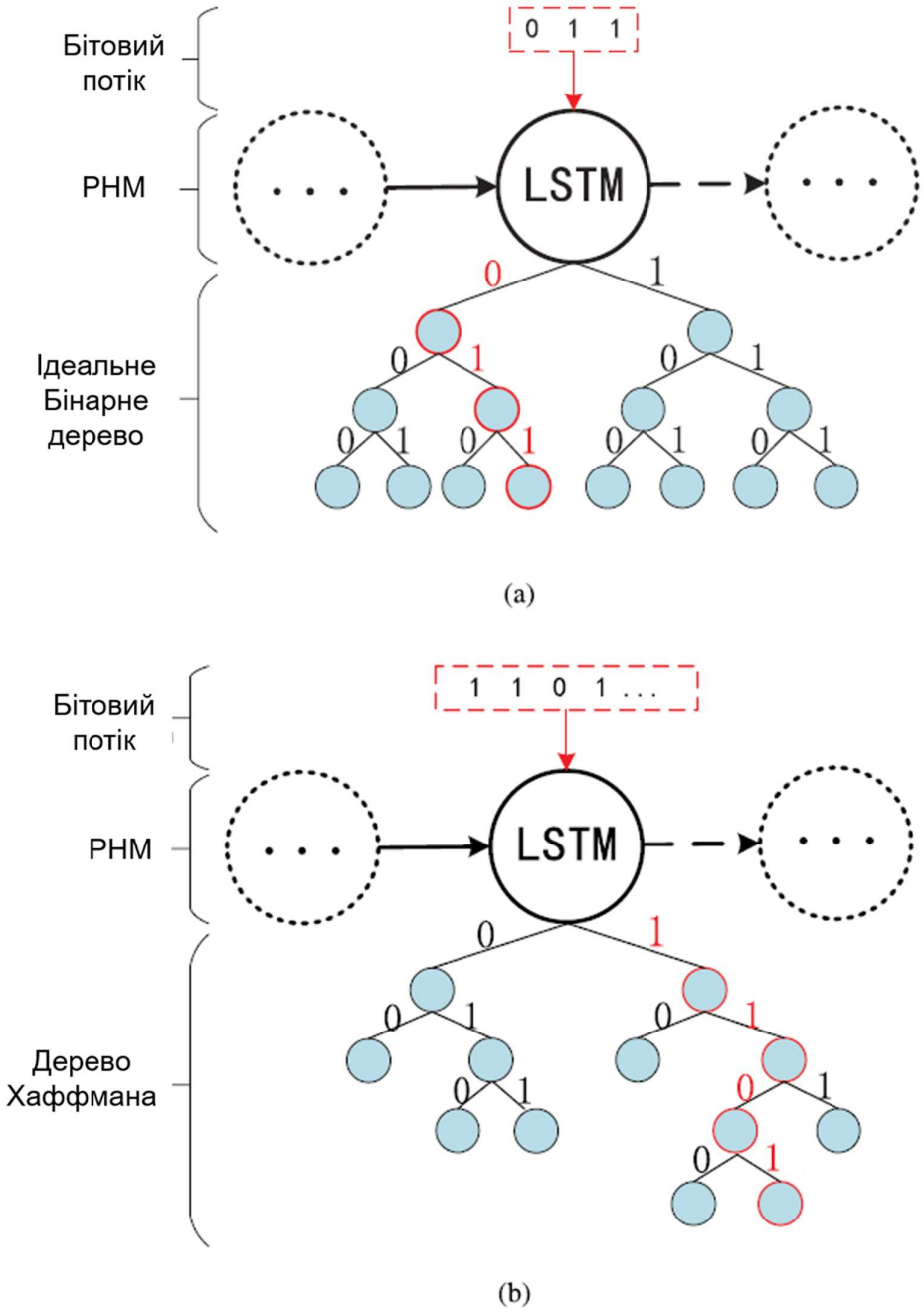

Рисунок 4.2 – Кодування слів у пулі кандидатів за допомогою (a) кодування фіксованої довжини (FLC) на основі ідеального двійкового дерева, або (b) кодування змінної довжини (VLC) на основі дерева Хаффмана.

Порівнюючи ці два методи кодування, переваги FLC полягають у простоті та ефективності, а довжина коду кожного слова є певною. Перевага VLC полягає в тому, що різниця в умовному розподілі ймовірності кожного слова повністю розглядається, що дозволяє легше вибирати слова з вищою ймовірністю, для того,

щоб якість сформованого тексту була краща. Але недолік полягає в тому, що потрібно будувати дерево Хаффмана на кожній ітерації процесу генерації, що впливає на ефективність генерації. В експериментальній частині ми проведемо детальне порівняння цих двох методів кодування.

Коли всі слова у пулі кандидатів закодовані, процес приховування інформації полягає у виборі відповідного листового вузлу, як виходу поточного часу згідно з потоком двійкового коду, який потрібно приховати. Перед приховуванням інформації спочатку слід визначити розмір пулу кандидатів (РПК) у кожен момент часу. Процес приховування інформації полягає в одночасному зчитуванні бітів довжини кожного слова, пошуку відповідного закодованого листового вузла від кореневого вузла ідеального бінарного дерева, та використовування його відповідного слова як вихід поточного часу для завершення процесу бітового приховування. В процесі приховування інформації за методом VLC, потік бітів, який потрібно приховати, послідовно читається в кожен момент, а потім здійснюється пошук з кореневого вузла дерево Хаффмана по порядку, поки він не зустріне листковий вузол. Після, відповідне слово є виходом в поточний час, і впровадження інформації завершено. Коли всю конфіденційна інформація вбудовано, під час кожної наступної ітерації, модель вибирає і виводить слово з найбільшою ймовірністю, щоб гарантувати якість сформованого речення.

Якщо представляти саме алгоритм приховування інформації, то вхідними даними будуть: таємний потік бітів, розмір пулу кандидатів, список ключових слів, який є необхідним для уникнення умови, що дві рівні послідовності бітів утворюють два еквівалентні текстові речення. Вихідним результатом буде згенерований стеганографічний текст. Алгоритм:

1) Підготовка даних та тренування РНМ;

2) Поки не закінчиться вхідний бітовий потік, поки не закінчиться кожне речення, обчислюється ймовірнісний розподіл наступного слова, відповідно до раніше сформованих слів, використовуючи навчену РНМ;

3) Зменшення ймовірності передбачення всіх слів та вибір найбільш сортованих слів для формування пулу кандидатів;

4) Побудування бінарного дерева або дерева Хаффмана згідно з розподілом ймовірності кожного слова в ПК та кодування дерева;

5) Читання двійкового потоку та пошук з кореня дерева відповідно до правил кодування доки відповідний листовий вузол знайдено і виведено відповідне йому слово;

6) Якщо кінець речення, вибирається випадкове ключове слово з списку, як початок нового речення;

7) Поки не кінець поточного речення, вибирається слово з найвищою ймовірністю з пулу кандидатів, в якості виходу на поточний час;

8) Продовжувати вибирати слова на кожний момент часу, доки не закінчиться останнє речення;

9) Повернути сформовані речення.

За допомогою цього методу можна генерувати велику кількість непримітних природних речень відповідно до вхідного секретного потоку бітів. А потім ці сформовані тексти можуть бути відправлені через відкритий канал для досягнення мети приховування та надсилання конфіденційної інформації з високою прихованістю.

4.2.3 Алгоритм вилучення інформації

Приховування та вилучення інформації - це дві цілком протилежні операції. Отримавши передане речення, одержувач повинен правильно декодувати конфіденційну інформацію, що міститься в ньому. Процес приховування інформації і видобутку в основному однаковий. Також необхідно використовувати РНМ для обчислення умовної ймовірності розподілу кожного слова в кожен момент, потім будувати той же пул кандидатів і використовувати той самий метод кодування для кодування слів у ПК.

Отримавши тексти, які містять кілька речень, одержувач вводить перше слово кожного речення як ключ в РНМ, який буде обчислювати ймовірність розподілу слів у кожен наступний момент часу по черзі. На кожному моменті часу, коли одержувач отримує розподіл ймовірностей поточного слова, він спочатку сортує всі слова у словнику в порядку спадання ймовірності та вибирає верхні

слова для формування пулу кандидатів. Потім він будує ідеальне бінарне дерево або дерево Хаффмана відповідно до тих самих правил кодування слів в ПК. Нарешті, згідно з фактично переданим словом в поточний момент, шлях від відповідного листовий вузла до кореневого вузлу визначається, так що можна успішно і точно декодувати приховані біти в поточному слові. Таким чином, бітовий потік, прихований у оригінальні тексти, можна витягати дуже швидко і без помилок.

Вхідними даними алгоритму є сформовані речення та розмір пулу кандидатів, вихідними – таємний потік бітів. Алгоритм:

1) Для кожного речення в тексті: ввести перше слово речення в навчену РНМ в якості ключа;

2) Для кожного слова речення: обчислюється ймовірнісний розподіл наступного слова, відповідно до попередніх слів, використовуючи РНМ;

3) Зменшення ймовірності передбачення всіх слів та вибір найбільш сортованих слів для формування пулу кандидатів;

4) Використання кодування фіксованої довжини або кодування змінної довжини для кодування слів у пулі кандидатів;

5) Якщо слово є у ПК: на основі фактично прийнятого слова в кожен момент, визначається шлях від кореневого вузлу до листового;

6) Відповідно до правил кодування дерева витягається відповідний біт та додається до потоку бітів;

7) Якщо слова нема у ПК: процес вилучення закінчується;

8) Отримання вилученого бітового потоку.

4.2.4 Експериментальні дослідження та аналіз

Було проведено декілька експериментів для тестування моделі з точки зору ефективності приховування інформації, непомітності інформації та ємності для приховування. Для ефективності приховування інформації був протестований середній час, необхідний для формування тексту фіксованою довжини та приховання потоку бітів фіксованою довжини. Для непомітності було порівняно та проаналізовано якість текстів, створених з різною швидкістю вбудовування, та

здатність протистояти стеганографічному виявленню. Для ємності для приховування було проаналізовано кількість інформації, яка може бути прихована в сформованих текстах, і порівняно з іншими стеганографічними алгоритмами.

Для того щоб модель могла автоматично імітувати та вивчати речення, написані людиною, необхідно мати велику кількість текстів, які написали люди, для тренування моделі, після чого можна отримати досить подібну статистичну мовну модель. Припускається, що відправник та отримувач передають інформацію через публічний канал. Тому, було вибрано три широкомасштабних текстових набори даних як тренувальні набори, а саме огляди фільмів, новини та твіттер. Для Twitter було обрано набір даних sentiment140, опублікований Алеком Го та ін. Він містить 1 600 000 твітів, отриманих за допомогою Twitter API. Для набору даних огляду фільмів ми вибрали широко використовуваний набір даних IMDB, опублікований Maas et al. Тексти двох наведених вище наборів даних належать до типу соціальних мереж. Крім того, також обрано набір даних новин, що містить відносно більше стандартних текстів для навчання моделі. Він містить 143 000 статей із 15 американських видань, у тому числі New York Times, Breitbart, CNN тощо. Теми набору даних переважно пов’язані з політикою.

Перед навчанням моделі потрібно провести попередню обробку даних, яка в основному складається з перетворення всіх слів у малі літери, видалення спеціальних символів, смайликів, веб-посилань та фільтрації низькочастотних слів. Деталі текстового набору після попередньої обробки представлені в таблиці 4.1.

Таблиця 4.1 – Деталі бази даних

	Середня довжина	Кількість речень	Кількість слів	Кількість унікальних
IMDB	18, 87	1,190,945	23,433,456	45,754
Twitter	9,43	2,598,986	24,876,432	43,382
Новини	21,98	1,809,456	46,234,964	40,987

Під час навчання моделі, з метою посилення регуляризації та запобігання переобладнання, застосовується механізм відсіву під час тренувального процесу. Використовується метод оптимізації [8].

Щодо результатів оцінки параметрів: ефективність приховування інформації обчислює час, необхідний для моделі щоб приховати фіксовану довжину конфіденціальної інформації. Оскільки в моделі слова в пулі кандидатів потрібно кодувати в дерево на кожній ітерації, розмір пулу кандидатів буде значно впливати на ефективність приховування. В таблиці 4.2 представлено середній час для кожної моделі для сформування речення з 50 слів з різним розміром пулу кандидатів.

Таблиця 4.2 – Середній час для кожної моделі для сформування речення з 50 слів з різним розміром пулу кандидатів

Розмір пулу кандидатів	FLC	VLC
2	3.987	4.587
4	4.576	8.234
8	4.668	14.124
16	4.769	25.692
32	4.829	47.056

З даних таблиці видно, що для моделі, яка використовує дерево Хаффмана, при зростанні розміру пулу кандидатів, значно зростає час, необхідний для генерації тексту. Це займає куди більше часу ніж побудова бінарного дерева на кожній ітерації. Тобто, VLC займає більше часу ніж FLC для побудови тексту певної довжини. В такому порівнянні, модель FLC має очевидні переваги в ефективності приховування інформації.

Щодо аналізу непомітності приховування. Це є найважливішим фактором оцінки системи приховування. Очікувано, що стеганографічна операція не спричинить відмінностей у розподіліносіїв у семантичному просторі. Модель, запропонована в роботі, автоматично генерує стеганографічний носій на основі конфіденціальної інфомації, не отримуючи носій наперед. Однак, також необхідно дотримуватися того, що згенерований текстовий носій повинен бути максимально відповідним до статистичного розподілу звичайного носія, що є куди складнішим.

Для тестування цього аспекту моделі чи є сфомовані речення близькими до нестеганографічних носіїв статистичної мовної моделі, інакше було б дуже легко відрізнити. Перплексивність – вимірювання (в теорії інформації) того, наскільки модель ймовірнісного розподілу або ймовірнісна модель передбачає вибірку. Це можна використати для порівняння моделей ймовірності. Перплексивність насправді обчислює різницю в статистичному розподілі мовної моделі між сформованими текстами та навчальними текстами.

В ході експерименту, кожна модель було протренеровано на обраній базі даних. Результати продемонстровано в таблиці 4.3.

Таблиця 4.3 – Відхід результату персплексивності різних методів стеганографії на основі бази даних

Метод		На основі LSTM		
	біти/слово	FLC	VLC	біти/слово (VLC)
IMDB	1	20.21	20.25	1.000
	2	29.96	24.03	1.878
	3	49.98	27.99	2.511
	4	99.04	36.35	3.165
	5	212.42	47.78	3.786
Twitter	1	19.12	21.07	1.000
	2	32.11	23.95	1.846
	3	53.96	30.56	2.487
	4	102.78	42.33	3.245
	5	214.98	54.87	3.871
Новини	1	18.32	18.56	1.000
	2	51.23	35.78	1.912
	3	80.89	40.76	2.335
	4	135.76	56.23	3.156
	5	273.76	65.98	3.897

Оскільки кількість вбудованих бітів на слово в VLC не є певною, було обчислено середню кількість бітів, прихованих в кожному слові сформованого тексту, що продемонстровано в самому правому стовпці таблиці 3. Можна зробити висновок, що для кожного стеганографічного алгоритму при збільшенні швидкості вбудовування зростає й персплективність. Тобто, різниця статистичного розподілу мови між сформованим текстом та навчальними зразками значно зростає. Причиною такого є те, що кількість бітів, що приховується в кожному слові, збільшується, слово, обране як вихід, все більше керується прихованими бітами в кожному ітеративному процесі, і все складніше обрати слова, що краще за всіх відповідають статистичному розподілу навчального тексту. Незалежно від того, скільки слів обрано в якості кандидатів, персплектвність сформованого тексту буде

лишатися на високому рівні. При підборі статистичної мовної моделі навчального тексту, метод автоматичної генерації тексту на основі LSTM має кращу продуктивність, оскільки існує відсутність подвійного наближення статистичної мовної моделі. Отже, створені тексти також більш відповідають статистичному розподілу навчальних текстів.

Для розглянутої моделі, оскільки умовна ймовірність розподілу повністю розглядається та кожне слово динамічно кодується, текст, який генерується, є більш відповідним до закону статистичного розподілу навчального тексту.

Отже, порівнюючи двох методів кодування, а саме FLC та VLC, можна зробити висновок, що ефективність кодування інформації VLC є нижчою ніж FLC, а якість сформованого VLC тексту є кращою. Основна причина полягає в тому, що метод FLC ігнорує різницю в умовному ймовірнісному розподілі кожного слова в пулі кандидатів під час процесу кодування. Метод VLC повністю використовує різницю в умовному ймовірнісному розподілі кожного слова в пулі кандидатів. Він виконує кодування Хаффмана відповідно до різних умовних ймовірностей кожного слова, тому слова, які більш відповідні до мовної моделі, мають коротше кодування і їх легше вибрати в якості вихідних даних на кожній ітерації. Це робить текст, що формується, ближчим до статистичного розподілу навчального тексту.

Крім порівняння відмінностей у статистичних мовних моделях кожної згенерованої моделі, далі було порівняно загальну статистичну схожість розподілу між стеганографічним текстом, сформованим кожною моделлю, та навчальними зразками. Було використано метод вилучення багатовимірних семантичних особливостей тексту запропонований Q. Le та T. Mikolov [9]. Велика кількість експериментів довела, що розподіл тексту в багатовимірному просторі може відображати його семантичну актуальність. Рисунки 4.1 та 4.2 демонструють результати зменшення розмірності та візуалізація за допомогою техніки розподіленого стохастичного вбудовування сусідів (англ. t-Distributed Stochastic Neighbor Embedding (t-SNE)) [14].

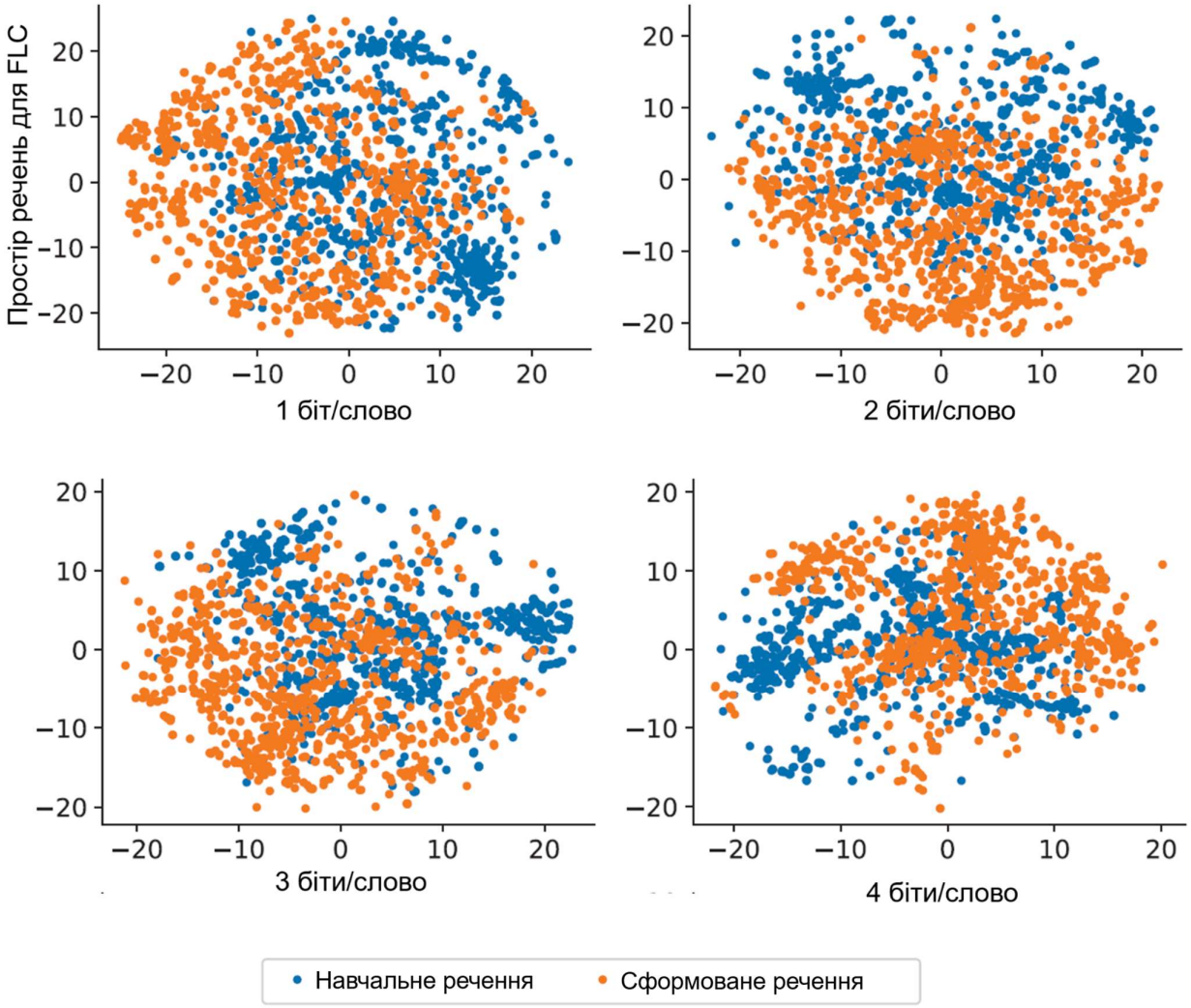

Рисунок 4.3 – Різниця у розподілі між сформованим стеганографічним текстом та навчальним текстом у семантичному просторі за різними швидкостями вбудовування (FLC).

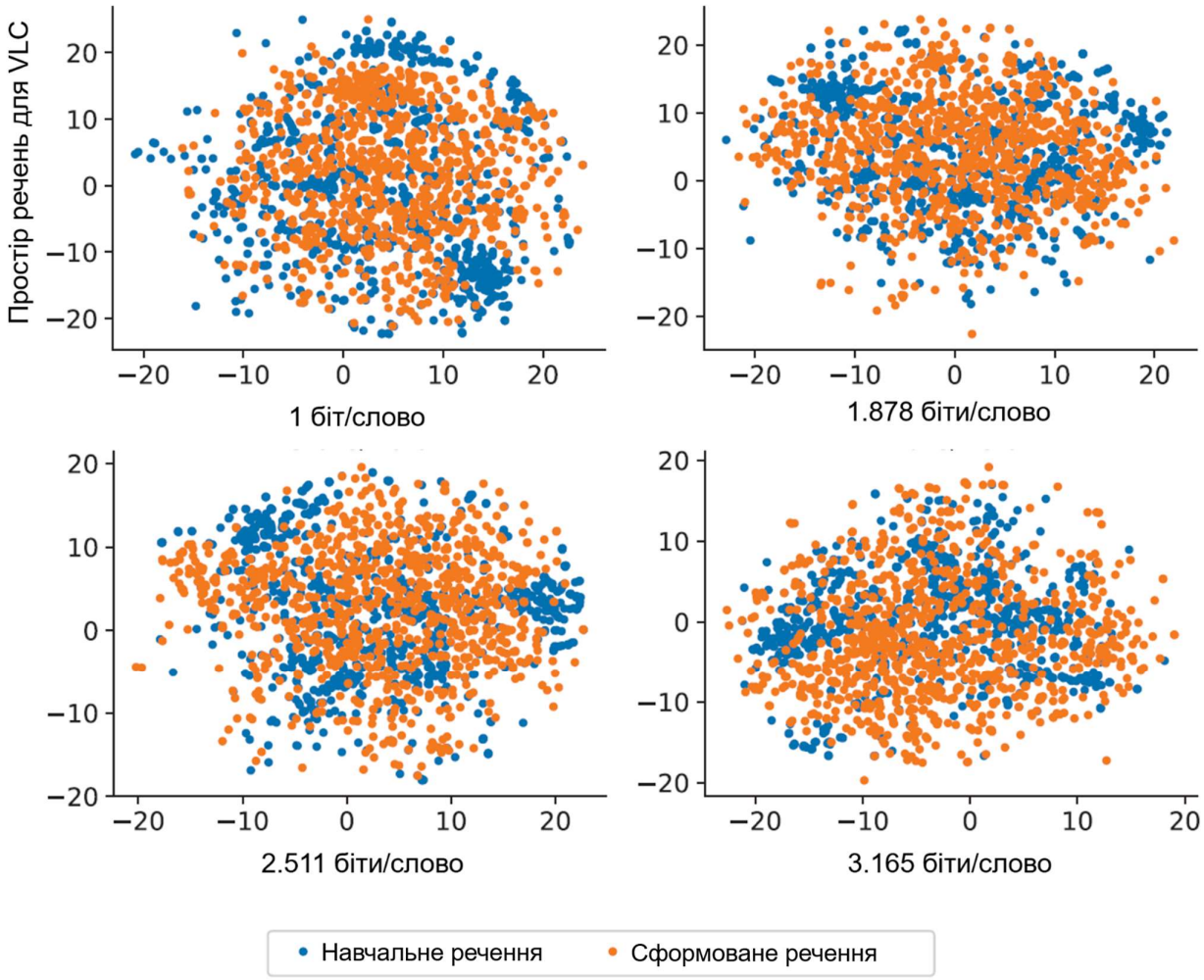

Рисунок 4.4 – Різниця у розподілі між сформованим стеганографічним текстом та навчальним текстом у семантичному просторі за різними швидкостями вбудовування (VLC).

Як видно з рисунків, є деякі відхилення в моделі, загальний розподіл все ще знаходиться в тій же самій області що й навчальний текст. Особливо для моделі VLC, навіть якщо швидкість вбудовування збільшується розподіл стеганографічного тексту та навчального тексту багато в чому збігаються.

Для кількісного порівняння різних методів, було розраховано відстань Вассерштайна сформованого тексту для кожного алгоритму та навчального тексту у багатовимірному семантичному просторі за різними швидкостями вбудовування [16]. У статистиці, відстань Вассерштайна, також відома як Earth Mover’s Distance (EMD), є мірою відстані між двома ймовірнісними розподілами регіону. Результати тестування представленні в таблиці 4.4.

Таблиця 4.4 – EMD для сформованого стеганографічного тексту і навчального тексту у багатовимірному семантичному просторі за різними швидкостями вбудовування

	На основі LSTM			
База даних	біти/слово	FLC	VLC	біти/слово
IMDB	1	1.799	1.812	1.000
	2	1.901	1.833	1.878
	3	2.102	1.871	2.511
	4	2.165	1.916	3.165
	5	2.189	1.941	3.786
Twitter	1	1.312	1.245	1.000
	2	1.389	1.256	1.876
	3	1.589	1.278	2.567
	4	1.678	1.267	3.234
	5	1.934	1.254	3.899
Новини	1	1.989	1.987	1.000
	2	2.067	2.039	1.765
	3	2.198	2.098	2.456
	4	2.345	2.187	3.345
	5	2.399	2.198	3.887

З даних таблиці 4.4 можна зробити висновок, що стеганографічний текст, згенерований запропонованою моделлю є близьким до навчального тексту в багатовимірному семантичному просторовому розподілі.

Для подальшого порівняння приховування кожного з стеганографічних алгоритмів, були протестовані та порівняні можливості кожного методу протистояти існуючим методам стегоаналізу. Спочатку, було випадково сформовано тренувальні зразки та для тестування, після чого було проведено імітоване виявлення атаки на цю вибірку, використовуючи метод стегоаналізу, який було запропоновано P. Meng [10]. Потім, було порівняно пропорції

стеганографічного тексту, що генерується кожним алгоритмом, які правильно виявляються. Результати показані в таблиці 4.5.

Таблиця 4.5 – Пропорції стеганографічного тексту, що генерується кожним алгоритмом, які правильно виявляються методом стеганографічного аналізу

	На основі LSTM			
База даних	біти/слово	FLC	VLC	біти/слово
IMDB	1	4.5%	8.0%	1.000
	2	6.0%	8.5%	1.878
	3	5.5%	5.0%	2.511
	4	6.5%	6.0%	3.165
	5	20.0%	6.5%	3.786
Twitter	1	5.5%	6.5%	1.000
	2	8.5%	5.5%	1.876
	3	13.5%	9.5%	2.567
	4	13.5%	8.0%	3.234
	5	19.0%	11.0%	3.899
Новини	1	9.0%	6.0%	1.000
	2	12.5%	10.0%	1.765
	3	16.5%	9.5%	2.456
	4	25.5%	12.0%	3.345
	5	25.0%	7.5%	3.887

Згідно з результатами таблиці 4.5, можна чітко побачити, що з двох моделей, запропонованих в роботі, саме VLC показує сильнішу непомітність інформації. В усіх випадках, пропорція виявлених зразків є меншою ніж десять відсотків.

Крім того, додатково було проведено тестування моделі на можливість протистояти стегоаналізу при високих швидкостях вбудовування (більш ніж 3 біти/слово). Було реалізовано три методи стегоаналізу, а саме методи запропоновані: 1) P. Meng [10]; 2) Z. Chen et al [11]; 3) S. Samanta, S. Dutta, and G.

Sanyal [12]. В цих методах, перший обчислив статистичну кореляцію, яка вимірюється взаємною інформацією, слів в сформованому стеганографічному тексті, і потім, використовуючи метод опорних методів, класифікував даний текст у стеганографічний текст або звичайний текст. Третій метод стеганографічного аналізу пропонує статистичні засоби стегоаналізу, на основі Басієвської оцінки та методології коефіцієнту кореляції. Додатково до вищезгаданих методів стегоаналізу, також було впроваджено поточний стан класифікатору тексту, який був запропонований A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov, для тестування можливості розрізняння сформованого стеганографічного тексту із саме навчального тексту.

Таблиця 4.6 фіксує точність виявлення трьох методів стегоаналізу для різних алгоритмів стеганографії при різної швидкості приховування.

Таблиця 4.6 – Результати різних методів стеганоаналізу

	Стегоаналіз	біт/слово	FLC	VLC	біт/слово (VLC)
IMDB	P. Meng, L. Hang, W. Yang, Z. Chen, and H. Zheng	3	0.456	0.476	2.511
		4	0.471	0.487	3.165
		5	0.471	0.479	3.786
	Z. Chen et al	3	0.551	0.563	2.511
		4	0.576	0.569	3.165
		5	0.615	0.590	3.786
	S. Samanta, S. Dutta, and G. Sanyal	3	0.554	0.530	2.511
		4	0.578	0.535	3.165
		5	0.623	0.547	3.786
	A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov	3	0.642	0.528	2.511
		4	0.728	0.584	3.165
		5	0.729	0.617	3.786

Продовження таблиці 4.6 - Результати різних методів стеганоаналізу

	Стегоаналіз	біт/слово	FLC	VLC	біт/слово (VLC)
Twitter	P. Meng, L. Hang, W. Yang, Z. Chen, and H. Zheng	3	0.453	0.478	2.511
		4	0.466	0.489	3.165
		5	0.467	0.478	3.786
	Z. Chen et al	3	0.556	0.567	2.511
		4	0.577	0.570	3.165
		5	0.623	0.594	3.786
	S. Samanta, S. Dutta, and G. Sanyal	3	0.567	0.534	2.511
		4	0.579	0.536	3.165
		5	0.634	0.551	3.786
	A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov	3	0.651	0.529	2.511
		4	0.731	0.586	3.165
		5	0.735	0.619	3.786
Новини	P. Meng, L. Hang, W. Yang, Z. Chen, and H. Zheng	3	0.465	0.479	2.511
		4	0.476	0.489	3.165
		5	0.476	0.481	3.786
	Z. Chen et al	3	0.554	0.566	2.511
		4	0.578	0.571	3.165
		5	0.623	0.595	3.786
	S. Samanta, S. Dutta, and G. Sanyal	3	0.565	0.587	2.511
		4	0.579	0.538	3.165
		5	0.633	0.549	3.786
	A. Joulin, E. Grave, P. Bojanowski, and T. Mikolov	3	0.647	0.531	2.511
		4	0.734	0.589	3.165
		5	0.737	0.619	3.786

З таблиці 4.6 можна зробити аналогічний висновок з попередніх експериментів, що здатність моделі VLC протистояти виявленню стегоаналізом сильніше ніж у FLC моделі.

Важливо зазначити, що хоча певний відсоток стеганографічних текстів, сформованих запропонованою моделлю, були правильно розрізнені при зустрічі з поточним станом класифікатору тексту, пропорція визнання все ж нижча за попередній метод.

Щодо аналізу ємності для приховування. Швидкість вбудовування обчислює кількість інформації, яку можна вбудувати в тексти, що є важливим показником для оцінки ефективності стеганографічного алгоритму. Це утворює протилежні відносини з приховуванням, яке зазвичай зменшується, коли швидкість вбудовування зростає. В ході роботи була протестована та проаналізована ємність для вбудовування запропонованої моделі.

Метод розрахунку коефіцієнта вбудовування полягає в поділі фактичної кількості вбудованих бітів за кількістю бітів, зайнятих всім сформованим текстом в комп'ютері. Математичний вираз можна представити як:

$$ER = \frac{1}{N}\sum_{i=1}^{N}\frac{(L_i - 1)\cdot k}{B(s_i)} = \frac{1}{N}\sum_{i=1}^{N}\frac{(L_i - 1)\cdot k}{8 \times \sum_{j=1}^{L_j} m_{i,j}} = \frac{(\bar{L} - 1)\times k}{8 \times \bar{L} \times \bar{m}} \qquad (4.2)$$

де N – кількість сформованих речень,

L_i – довжина i-го речення.

Параметр k вказує на кількість бітів, які вбудовані в кожне слово, $B(s_i)$ вказує на кількість бітів, зайнятих i-тим реченням в комп'ютері. Оскільки кожна англійська літера займає один байт в комп'ютері, тобто вісім бітів, кількість бітів, зайнятих кожним англійським реченням, дорівнює величині $B(s_i)$, де $m_{i,j}$ представляє кількість літер, що належать до j-го слова i-го речення. $\bar{L}$ та $\bar{m}$ представляють собою середню довжину кожного речення в сформованому тексті та середню кількість літер, що містяться в кожному слові.

Можна зробити висновок, що коли довжина сформованого тексту є постійною, швидкість вбудовування лінійно збільшується зі збільшенням кількості

бітів, що вбудовуються, на слово. Коли кількість бітів, вбудованих в кожне слово речення, є постійною величиною, тоді швидкість вбудовування зростає зі збільшенням довжини сформованих текстів. Але є верхня межа, максимальна швидкість вбудовування може досягати більше 20%.

В таблиці 4.7 представлені результати моделі, коли в слово вбудовуються 3, 5 та 7 бітів.

Таблиця 4.7 – Результати моделі

біти/слово	Швидкість вбудовування (%)
3	7.56
5	12.25
7	17.13

З даних таблиці чітко видно, що запропонована модель демонструє майже ідеальну швидкість вбудовування. Швидкість вбудовування більшості інших моделей доходить тільки до 1%.

Для прикладу, можна порівняти результати з методом, запропонованим T. Y. Liu and W. H. Tsai, в якому швидкість вбудовування досягла 5.4%. Така відмінність з'явилася тому, що цей метод використовував китайські літері, коли інші використовували англійські. Різниця між мовами полягає в тому, що кожне китайське слово займає 16 бітів в комп'ютері, коли для англійської мови, кожне слово в середньому складається з 4.8 літер, а кожна літера займає 8 бітів. Тобто, кожне англійське слово займає більшу кількість бітів ніж китайське слово. До того ж, коли китайські тексти використовуються як контейнер для вбудовування, середня швидкість вбудовування є значно вищою за використання англійських текстів як вектор вбудовування.

Запропонована модель також використовує англійські тексти як контейнер для вбудовування, але все одно її швидкість вбудовування все одно значно перевершує інші методи. Якщо спробувати використати китайську мову, то можна отримати навіть більшу швидкість вбудовування.

4.3 Обґрунтування рекомендацій щодо практичного застосування отриманих результатів

Лінгвістична стеганографія на основі технології автогенерації носія тексту - актуальна тема з великими перспективами та викликами. В роботі запропоновано використання лінгвістичної стеганографії на основі рекурентної нейронної мережі, які можуть автоматично генерувати високоякісні текстові «прикриття» для секретного потоку бітів, який потрібно приховати. Запропонований алгоритм можна вважати одним з найкращих в цій сфері, бо результати експериментальних випробувань показують, що запропонована в роботі модель перевершує всі попередні пов’язані з цим методи.

ВИСНОВКИ

У роботі представлено структуроване та всебічне уявлення про методи глибинного навчання, включаючи таксономію, що розглядає різні типи завдань реального світу, як контрольовані так і неконтрольовані. У таксономії беруться до уваги глибинні мережі для контрольованого або дискримінаційного навчання, неконтрольованого або генеративного навчання, а також гібридне навчання та відповідні інші. Узагальнюються реальні сфери застосування, де можна використовувати методи глибинного навчання.

На сьогодні, обробка природної мови є актуальним напрямком розвитку глибинного навчання, бо є необхідність обробляти велику кількість текстової інформації, структурувати, упорядковувати, систематизувати її.

Глибинне навчання являє собою актуальну сферу наукового знання, яка інтенсивно розвивається та має дуже значні перспективи. Одним з найперспективніших сучасних технологій глибинного навчання є застосування рекурентних нейронних мереж. Таким чином, рекурентні нейронні мережі можуть успішно застосовуватись для здійснення обробки мовної інформації. Нейронні мережі можуть виконувати ту роботу, для якої раніше потрібно було застосовувати велику кількість фахівців в області лінгвістики.

В роботі запропонована лінгвістична стеганографія з використанням рекурентної нейронної мережі, яка може автоматично генерувати текстові контейнери високої якості для приховування таємного потоку бітів. Модель тренувалася великою кількістю штучно створених зразків, після чого було отримано хорошу оцінку статистичної мовної моделі. В процесі генерації було запропоновано кодування фіксованої довжини та кодування змінної величини для кодування слів на основі їхнього умовного розподілу ймовірності. Декілька експериментів було проведено для тестування моделі щодо ефективності вбудовування, непомітності вбудовування, ємності для вбудовування. Результати показали, що запропонована модель перевершує всі зв’язані методи.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Karhunen J, Raiko T, Cho KH. Unsupervised deep learning: a short review. In: Advances in independent component analysis and learning machines. 2015; P. 125–142.
2. Sarker IH. Deep cybersecurity: a comprehensive overview from neural network and deep learning perspective. SN Computer. Science. 2021; P. 1–16.
3. Machine learning and deep learning methods for cybersecurity / Xin Y та ін., 2018; P. 65–81.
4. Sarker IH. Machine learning: Algorithms, real-world applications and research directions. 2021; P. 1–21.
5. Sarker IH, Furhad MH, Nowrozy R. Ai-driven cybersecurity: an overview, security intelligence modeling and research directions. 2021; P. 1–18.
6. Sarker IH. Data science and analytics: an overview from data-driven smart computing, decision-making and applications perspective. 2021.
7. LeCun Y, Bengio Y, Hinton G. Deep learning. 2015; P. 436–444.
8. Deng L. A tutorial survey of architectures, algorithms, and applications for deep learning. 2014; 3 p.
9. Generative adversarial nets / Goodfellow I. та ін. 2014; P. 2672–2680.
10. Kohonen T. The self-organizing map. 1990; P. 1464–1480.
11. Inductive principles for restricted boltzmann machine learning / Marlin B. та ін. 2010; P. 509–516.
12. Hinton GE. Deep belief networks. 2009.
13. Bennet K. Linguistic Steganography: Survey, analysis and robustness concerns for hiding information in text. 2004; 120 p.
14. Глибовець М.М., Олецький О.В. Системи штучного інтелекту. Київ: *«КМ Академія»*, 2002.
15. Memory Networks. URL: https://gist.github.com/shagunsodhani/c7a03a47b3d709e7c592fa7011b0f33e (дата звернення: 02.11.2022).

16. Word2Vec. URL: https://skymind.ai/wiki/word2vec (дата звернення 02.11.2022).
17. A Gentle Introduction to Neural Machine Translation. URL: https://machinelearningmastery.com/introductionneural-machine-translation (дата звернення: 07.10.2022).
18. Learning word vectors for sentiment analysis / A. L. Maas та ін. 2011; P. 142–150.
19. D. P. Kingma, J. Ba. Adam. A method for stochastic optimization. URL: https://arxiv.org/abs/1412.6980 (дата звернення 07.10.2022).
20. Q. Le and T. Mikolov. Distributed representations of sentences and documents. 2014; P. 1188–1196.
21. Linguistic steganography detection algorithm using statistical language model / P. Meng та ін. 2009; P. 540–543.
22. Z. Chen. Linguistic steganography detection using statistical characteristics of correlations between words. 2008; P. 224–235.
23. S. Samanta, S. Dutta, G. Sanyal. A real time text steganalysis by using statistical method. 2016; P. 264–268.
24. T. Y. Liu, W. H. Tsai. A new steganographic method for data hiding in Microsoft word documents by a change tracking technique. 2007; P. 24–30.
25. L. V. D. Maaten. Accelerating t-SNE using tree-based algorithms. 2014; P. 3221–3245.
26. D. A. Huffman, A method for the construction of minimum-redundancy codes. 1952; P. 1098–1101.
27. Y. Rubner, C. Tomasi, and L. J. Guibas. The earth mover's distance as a metric for image retrieval. 2000; P. 99–121.
28. Шаров С.В., Лубко Д.В., Осадчий В.В. Інтелектуальні інформаційні системи. Мелітополь, 2015.
29. Методы обработки и анализа текстов на природных языках / Доценко С. И. та ін. 2018.
30. Черемський І.А., Черних О.П. Галузі застосування рекурентних нейронних мереж, Харків, 2018.

31. Шеремет О. І., Запорожець В. С. Застосування рекурентних нейронних мереж для виконання машинного рерайту. 2018.
32. A set of examples around pytorch in Vision, Text, Reinforcement Learning, URL: https://github.com/pytorch/examples (дата звернення: 07.11.2022).

ДОДАТОК А

Лістинг програми для власної реалізації TensorFlow:

```
import tensorflow as tf
import HuffmanEncoding
import Model
import random
import sys
import Config_movie as Config
import numpy as np
import collections
import os

bitnum = sys.arv[1]
bitnum = np.int32(bit_num)
index = sys.arv[2]

os.environ["DEVICES"] = "0"

configtf = tf.ConfigProto()
configtf.intra_op_paralelism_threads = 1
configtf.inter_op_paralelism_threads = 1
configtf.gpu_options.alow_growth = True

wordall = []
config = Config.Config()
data = open(file, 'r').readlines()
file = 'movie.txt'


data_size, _vocab_size = len(wordall), len(words)
wordtoidx = {wo: i for i, wo in enumerate(words)}
idxtoword = {i: wo for i, wo in enumerate(words)}


config.vocabsize = len(word_to_idx)
len_of_generation = config.len_of_generation

idx_unknown = wordtoidx['unknown']

def runepoch(session, m, data, eval_op, state=None):
    x = data.reshape((1, 1))
    prob, _state, _ = session.run([m._prob, m.final_state, eval_op],
{m.input_data: x, m.initial_state: state})
    return prob, _state

def prostartword(statistics1):
    sel_word_sta = []
    sel_value_sta = []
    for i in range(100):
        k = statistics1[i]
        key = k[0]
        value = k[1]
        sel_word_sta.append(key)
        sel_value_sta.append(value)
    sel_value_sta = np.array(sel_value_sta)
    sel_value_sta = sel_value_sta/float(sum(sel_value_sta))
```

```
    start = np.random.choice(sel_word_sta, 1, p=sel_value_sta)
    start_word = start[0]
    while not start_word.islower():
        start = np.random.choice(sel_word_sta, 1, p=sel_value_sta)
        startword = start[0]
    return startword

def main( ):
    os.makedirs('movie', exist_ok=True)
    with tf.Graph().as_default(), tf.Session(config=config_tf) as session:
        config.batch_size = 0
        config.num_steps = 0
        intlzr = tf.random_uniform_initializer(-config.init_scale,
config.init_scale)
        with tf.variable_scope("model", reuse=None, initializer=initializer):
            test = Model.Model(is_training=False, config=config)

        modelsaver = tf.train.Saver()
        print 'loading ...'
        model_saver.restore(session, config.model_path+'70')
        print 'Done!'

        startwords = []
        data = open('movie.txt', 'r').readlines()
        for i in range(len(data)):
            lines = data[i].strip()
            lines = line_ste.split(' ')
            startword = line_ste[0]
            startwords.append(start_word)
        statistics = collections.Counter(start_words)
        statistics1 = sorted(statistics.items(), key=lambda item: item[1],
reverse=True)

        bitstream = open('.bitstream.txt', 'r').readline()
        outfile = open('movie' + str(bit_num) + 'bit' + '_' + index + '.txt', 'w')
        bitfile = open('movie' + str(bit_num) + 'bit' + '_' + index + '.bit', 'w')
        bit_index = random.randint(0, 30000)
        count = 0
        while count < 100:
            startword = prostartword(statistics1)
            if startword == 'unk':
                continue

            startidx = word_to_idx[start_word]
            state = mtest.initial_state.eval()
            testdata = np.int32([start_idx])
            prob, state = run_epoch(session, mtest, test_data, tf.no_op(), _state)
            gen_res = [start_word]
            gen = word_to_idx['unk']
            while gen == word_to_idx['unk']:
                gen = np.random.choice(config.vocab_size, 1, p=prob.reshape(-1))
                gen = gen[0]
            test_data = np.int32(gen)
            gen_res.append(idx_to_word[gen])
            bit = ""

            for i in range(len_of_generation - 2):
                if idx_to_word[gen] in ['\n', '']:
                    break
                prob, _state = run_epoch(session, mtest, test_data, tf.no_op(),
state)
                p = prob.reshape(-1)
                p[idx_unknown] = 0
```

```
                probsort = sorted(p)
                probsort.reverse()
                wordprob = [prob_sort[i] for i in range(2**int(bit_num))]

                p = p.tolist()
                words_prob = [(p.index(word_prob[i]), word_prob[i]) for i in
range(2**int(bit_num))]
                nodes = HuffmanEncoding.createNodes([item[1] for item in
words_prob])
                root = HuffmanEncoding.createHuffmanTree(nodes)
                codes = Huffman_Encoding.huffmanEncoding(nodes, root)
                for i in range(2**int(bit_num)):
                    if bitstream[bit_index:bit_index+i+1] in codes:
                        code_index =
codes.index(bitstream[bit_index:bit_index+i+1])
                        gen = wordsprob[code_index][0]
                        test_data = np.int32(gen)
                        gen_res.append(idx_to_word[gen])
                        if idx_to_word[gen] in ['\n', '']:
                            break
                        bit += bit_stream[bit_index: bit_index+i+1]
                        bit_index = bit_index+i+1
                        break

            if len(genres) < 5:
                continue

            gensen = ' '.join([word for word in gen_res if word not in ["\n",
""]])
            count = count + 1
            outfile.write(gen_sen+"\n")
            bitfile.write(bit)


if __name__ == "__main__":
    tf.app.run()
```

ДОДАТОК Б

Лістинг програми для створення та тренування моделі

```
import argparse
import model
import time
import data
import math
import torch
import torch.onnx
import os
import torch.nn as nn


parser = argparse.ArgumentParser(description='LanguageModel')
parser.add_argument('--data', type=str, default='./data/',
                    help=' data corpus')
parser.add_argument('--model', type=str, default='LSTM',
                    help='type of recurrent net')
parser.add_argument('--emsize', type=int, default=200,
                    help='size of word embeddings')
parser.add_argument('--batch_size', type=int, default=20, metavar='N',
                    help='batch size')
parser.add_argument('--log-interval', type=int, default=200, metavar='N',
                    help='report interval')
parser.add_argument('--nlayers', type=int, default=2,
                    help='number of layers')
parser.add_argument('--bptt', type=int, default=35,
                    help='sequence length')
parser.add_argument('--lr', type=float, default=20,
                    help='initial learning rate')
parser.add_argument('--cuda', action='store_true',
                    help='use CUDA')
parser.add_argument('--epochs', type=int, default=40,
                    help='upper epoch limit')
parser.add_argument('--dropout', type=float, default=0.2,
                    help='dropout applied to layers (0 = no dropout)')
parser.add_argument('--tied', action='store_true',
                    help='the word embedding and softmax weights')
parser.add_argument('--seed', type=int, default=1111,
                    help='random seed')
parser.add_argument('--save', type=str, default='model.pt',
                    help='final model')
parser.add_argument('--nhead', type=int, default=2,
                    help='the number of heads in the encoder/decoder)
parser.add_argument('--onnx-export', type=str, default='',
                    help='export the final model')
parser.add_argument('--dry-run', action='store_true',
                    help='verify')

args = parser.parse_args()

torch.manual_seed(args.seed)
if torch.cuda.is_available():
    if not args.cuda:
        print("WARNING")

dvce = torch.device("cuda" if args.cuda else "cpu")
```

```
crps = data.Corpus(args.data)

eval_batch_size = 10
train_data = batchify(corpus.train, args.batch_size)
val_data = batchify(corpus.valid, eval_batch_size)
test_data = batchify(corpus.test, eval_batch_size)

def batchify(data, bsz):
    nbatch = data.size(0) // bsz

    data = data.narrow(0, 0, nbatch * bsz)

    data = data.view(bsz, -1).t().contiguous()
    return data.to(device)


ntokens = len(corpus.dictionary)
if args.model == 'Transformer':
    model = model.TransformerModel(ntokens, args.emsize, args.nhead, args.nhid,
args.nlayers, args.dropout).to(device)
else:
    model = model.RNNModel(args.model, ntokens, args.emsize, args.nhid,
args.nlayers, args.dropout, args.tied).to(device)

criterion = nn.NLLLoss()

def evaluate(data_source):

    model.eval()
    total_loss = 0.
    ntokens = len(corpus.dictionary)
    if args.model != 'Transformer':
        hidden = model.init_hidden(eval_batch_size)
    with torch.no_grad():

def repackagehidden(h):


    if isinstance(h, torch.Tensor):
        return h.detach()
    else:
        return tuple(repackagehidden(v) for v in h)



def get_batch(source, i):
    seq_len = min(args.bptt, len(source) - 1 - i)
    data = source[i:i+seq_len]
    target = source[i+1:i+1+seq_len].view(-1)
    return data, target



        for i in range(0, data_source.size(0) - 1, args.bptt):
            data, targets = get_batch(data_source, i)
            if args.model == 'Transformer':
                output = model(data)
                output = output.view(-1, ntokens)
            else:
                output, hidden = model(data, hidden)
                hidden = repackage_hidden(hidden)
            total_loss += len(data) * criterion(output, targets).item()
```

```
    return total_loss / (len(data_source) - 1)


def training():

    model.training()
    total_loss = 0.
    start_time = time.time()
    ntokens = len(corpus.dictionary)
    if args.model != 'Transformer':
        hidden = model.init_hidden(args.batch_size)
    for batch, i in enumerate(range(0, train_data.size(0) - 1, args.bptt)):
        data, targets = get_batch(train_data, i)

        model.zero_grad()
        if args.model == 'Transformer':
            output = model(data)
            output = output.view(-1, ntokens)
        else:
            hidden = repackage_hidden(hidden)
            output, hidden = model(data, hidden)
        loss = criterion(output, targets)
        loss.backward()


        torch.nn.utils.clip_grad_norm_(model.parameters(), args.clip)
        for p in model.parameters():
            p.data.add_(p.grad, alpha=-lr)

        total_loss += loss.item()

        if batch % args.log_interval == 0 and batch > 0:
            cur_loss = total_loss / args.log_interval
            elapsed = time.time() - start_time
            print(' '
                    '.format(
                epoch, batch, len(train_data) // args.bptt, lr,
                elapsed * 1000 / args.log_interval, cur_loss, math.exp(cur_loss)))
            total_loss = 0
            start_time = time.time()
        if args.dry_run:
            break


def export_onnx(path, batch_size, seq_len):

    model.eval()
    dmmy_input = torch.LongTensor(seq_len * batch_size).zero_().view(-1,
batch_size).to(device)
    hidden = model.init_hidden(batch_size)
    torch.onnx.export(model, (dummy_input, hidden), path)

lr = args.lr
best_val_loss = None

try:
    for epoch in range(1, args.epochs+1):
        epoch_start_time = time.time()
        train()
        val_loss = evaluate(valdata)
        print('-' * 89)
        print('| end of '.format(epoch, (time.time() - epoch_start_time),
                                           valloss, math.exp(val_loss)))
```

```
        print('-' * 98)

        if not bestvalloss or valloss < bestvalloss:
            with open(args.save, 'wb') as f:
                torch.save(model, f)
            bestvalloss = valloss
        else:.
            lr /= 5.0
except KeyboardInterrupt:
    print('-' * 98)
    print('Exiting)


with open(args.save, 'rb') as f:
    model = torch.load(f)

    if args.model in ['RNN_TANH', 'RNN_RELU', 'LSTM', 'GRU']:
        model.rnn.flatten_parameters()

testloss = evaluate(testdata)
print('=' * 89)
print('| End of training'.format(
    test_loss, math.exp(test_loss)))
print('=' * 98)

if len(args.onnx_export) > 0:

    export_onnx(args.onnx_export, batch_size=1, seq_len=args.bptt)
```

ДОДАТОК В

Лістинг програми для генерації нових зразків речень для мовної моделі

```
import argparse
import torch
import data

parser = argparse.ArgumentParser(description='LanguageModel')


parser.add_argument('--data', type=str, default='./,
                    help='location of the data corpus')
parser.add_argument('--checkpoint', type=str, default='./model.pt',
                    help='model checkpoint')
parser.add_argument('--out', type=str, default='gnrtd.txt',
                    help='output file for text')
parser.add_argument('--words', type=int, default='1000',
                    help='number of word')
parser.add_argument('--seed', type=int, default=1111,
                    help='random seed')
parser.add_argument('--cuda', action='store_true',
                    help='use CUDA')
parser.add_argument('--temperature', type=float, default=1.0,
                    help='temperature’)
parser.add_argument('--log-interval', type=int, default=100,
                    help='report')
args = parser.parse_args()


torch.manual_seed(args.seed)
if torch.cuda.is_available():
    if not args.cuda:
        print("WARNING")

device = torch.device("cuda" if args.cuda else "cpu")

if args.temperature < 1e-3:
    parser.error("--temperature has to be greater or equal 1e-3")

with open(args.checkpoint, 'rb') as f:
    model = torch.load(f).to(device)
model.eval()

corpus = data.Corpus(args.data)
ntokens = len(corpus.dictionary)

is_transformer_model = hasattr(model, 'model_type') and model.model_type ==
'Transformer'
if not is_transformer_model:
    hidden = model.init_hidden(1)
input = torch.randint(ntokens, (1, 1), dtype=torch.long).to(device)

with open(args.outf, 'w') as outf:
    with torch.no_grad():  # no tracking history
        for i in range(args.words):
            if is_transformer_model:
                output = model(input, False)
                word_weights = output[-
```

```
1].squeeze().div(args.temperature).exp().cpu()
                word_idx = torch.multinomial(word_weights, 1)[0]
                word_tensor = torch.Tensor([[word_idx]]).long().to(device)
                input = torch.cat([input, word_tensor], 0)
            else:
                output, hidden = model(input, hidden)
                word_weights = output.squeeze().div(args.temperature).exp().cpu()
                word_idx = torch.multinomial(word_weights, 1)[0]
                input.fill_(word_idx)

            word = corpus.dictionary.idx2word[word_idx]

            outf.write(word + ('\n' if i % 20 == 19 else ' '))

            if i % args.log_interval == 0:
                print('| Generated {}/{} words'.format(i, args.words))
```