МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Київський національний університет імені Тараса Шевченка
Навчально-науковий інститут філології
Катедра української мови та прикладної лінгвістики

СИМПЛІФІКАЦІЯ ТЕКСТІВ ЯК ОДИН ЗІ СПОСОБІВ АВТОМАТИЧНОЇ ОБРОБКИ ПРИРОДНОМОВНИХ ТЕКСТІВ

Кваліфікаційна робота
освітнього ступеня «бакалавр»
за спеціальністю 035 «Філологія»,
спеціалізацією 035.10 «Прикладна лінгвістика»,
галузі знань 03 «гуманітарні науки»
ОПП «Прикладна (комп'ютерна) лінгвістика та англійська мова»
студентки IV курсу
Ірини ПЛЕШИВЦЕВОЇ

Наукові керівники:
д.філол.н., проф. Наталія ДАРЧУК
к.техн.н., доц. Микола КОСТІКОВ

«Допущено до захисту»
Протокол № 11 засідання кафедри
української мови та прикладної лінгвістики
ННІФ від 01.06.2023
Завідувач кафедри_____**Сергій Різник**

КИЇВ – 2023

ЗМІСТ

ВСТУП

Актуальність роботи. Комп'ютеризація багатьох сфер діяльности людини є невід'ємним елементом сьогодення. Через нескінченний потік інформації сучасним людям усе складніше зосередитися на чомусь одному. Теперішні реалії вимагають від нас гнучкости, уміння швидко обробляти великі обсяги інформації та за потреби вчасно послуговуватися нею. Одним з нагальних питань, пов'язаних з інформацією, є її викладання. Важливо доносити її у зрозумілому для споживачів вигляді. Але з різних причин не всі люди можуть реалізувати свою потребу в отриманні зрозумілої інформації. Автоматична симпліфікація текстів могла би розв'язати цю проблему.

Метою роботи є написання програми, основу якої складатиме модель машинного навчання, що тренується здійснювати автоматичну симпліфікацію українськомовних текстів про тварин шляхом пошуку зв'язків між оригінальними статтями з Вікіпедії та їхніми спрощеними вручну варіантами.

Об'єкт дослідження – автоматична симпліфікація текстів як одне із завдань NLP.

Предметом дослідження є шляхи реалізації автоматичної симпліфікації природномовних текстів.

Для досягнення поставленої мети потрібно було виконати такі **завдання**:

1) дати визначення симпліфікації, описати цілі та загальні методи її здійснення;
2) дослідити та описати сучасні методи синтаксичної симпліфікації;

3) дослідити та описати актуальні методи лексичної симпліфікації;
4) окреслити основні етапи історії застосування штучних нейронних мереж у NLP;
5) описати класифікацію штучних нейронних мереж;
6) дослідити підходи здійснення реферування та генерування текстів, що можуть бути корисні при реалізації симпліфікації;
7) описати особливості створеної моделі спрощення текстів;
8) надати ідеї для підвищення ефективности роботи моделі;
9) зробити висновки.

Методи дослідження: описовий, аналітичний, порівняльний, прийом спостереження.

Наукова новизна здобутих результатів визначається тим, що створено модель автоматичної автоматичної симпліфікації текстів про тварин українською мовою, яка, виконуючи одне із завдань NLP, може стати здобутком в українській прикладній лінгвістиці.

Практичне значення роботи. Результати дослідження можуть бути використані для наближення швидкого доступу до інформації про тварин українською мовою для людей з низькими навичками читання та/або тих споживачів, що не є носіями української та/або експертами в зоології.

РОЗДІЛ 1

СИМПЛІФІКАЦІЯ: ЗАГАЛЬНА ІНФОРМАЦІЯ

1.1. Визначення, мета й основні підходи в симпліфікації

Симпліфікація – це процес синтаксичних та лексичних перетворень тексту, які спрямовані на його спрощення зі збереженням найголовнішої інформації. Основна мета симпліфікації – зробити текст простішим для розуміння людьми з низькими навичками читання [38]. З визначення можна зробити висновок, що спрощення синтаксису передбачає добір більш легких для розуміння конструкцій, а спрощення лексики – аналіз лексичного складу тексту та корегування його [лексичного складу] в разі наявности специфічної лексики, маловживаних слів, які можуть бути незрозумілими для більшости читачів.

Симпліфікація має дві основні цілі: спрощення тексту для людей і спрощення тексту для комп'ютера для подальшого розв'язання певних мовних завдань [28].

Хто може потребувати спрощеного тексту? Насамперед, ідеться про людей з вадами слуху. Через часткову або повну відсутність слуху вони змалечку не мають змоги розвивати когнітивні та лінгвістичні навички однаково зі своїми однолітками. Оскільки і синтаксичний, і семантичний аналізи тексту під час його читання обмежені робочою пам'яттю, то що більше робочої пам'яти потрібно на збереження інформації під час сприйняття тексту, то менше її лишається на формування в голові остаточної картини описаної в тексті ситуації. У результаті люди з вадами слуху стикаються з труднощами, читаючи складні із синтаксичної точки зору речення [36].

Хворі на афазію також мають проблеми зі сприйняттям тексту. Афазія – це мовне порушення, яке може виникнути внаслідок інфаркту, інсульту або нещасного випадку, що призводять до пошкодження головного мозку. Мовні труднощі в таких людей залежать від ступеня пошкодження та ділянки головного мозку. Загалом усі, хто має афазію, так чи інакше складно сприймають довгі речення, рідковживані слова, складні граматичні конструкції [36] [10].

Люди, що не є носіями мови, також можуть потребувати спрощеного тексту для швидкого розуміння його змісту [31]. Можна спрощувати текст з урахуванням рівня володіння мовою кожного конкретного іноземця з метою покращення знання цієї мови. Знаючи, що симпліфікація текстів спрямована на людей з низькими навичками читання, можемо припустити, що для дітей вона [симпліфікація текстів] означає легше сприйняття матеріалу на певну тему, збереження цікавости до неї, заохочення їх до подальшого її вивчення.

Підсумовуючи інформацію про цілі симпліфікації, можна сказати, що вона слугує для людей з не надто розвиненими навичками читання або тих, хто має деякі хвороби, які обмежують здатність аналізувати великі пласти інформації. Водночас симпліфікація змінює текст для подальшого розв’язання інших завдань, пов’язаних з обробкою природної мови, зокрема здобування інформації про семантичні зв’язки (Relation Extraction) [1], розмічування семантичних ролей (Semantic Role Labeling), машинний переклад (Machine Translation) [15], парсинг (Parsing), реферування (Summarization) [28], генерування питань до тексту [41].

Симпліфікація може здійснюватись або вручну, або автоматично. Між цими двома підходами є чимало відмінностей, кожен з них має свої переваги та недоліки. Спрощення вручну вирізняється індивідуальним підходом до кожного тексту, уважним добором доречних мовних засобів до

тієї чи іншої комунікативної ситуацій. При цьому воно вимагає значних фінансових витрат. Також слід враховувати, що швидкість появи в загальному доступі спрощених текстів має бути не меншою, ніж швидкість виходу новин, якщо йдеться про симпліфікацію медійних текстів. Такі медіа, як шведське 8 Sidor, норвезьке Klar Tale, бельгійські l'Essentiel та Wablie, датське Radio Ligetil, італійське Due Parole, фінське Selo-Uutiset, публікують новини, написані адаптованою, легкою для читання мовою. Також слід згадати про сайт Literacyworks, що подає новини від CNN в оригінальному та спрощеному варіантах [31].

Автоматична симпліфікація текстів здійснюється за допомогою програми, яка аналізує кожне речення і вирішує, чи є воно складним, і, якщо так, то проводить з ним певні дії, щоб спростити. Для того, щоб навчити програму аналізувати речення (ідеться про машинне навчання), потрібен корпус текстів, спрощених вручну. Для англійської мови у пригоді можуть стати пари текстів із сайту Literacyworks або тексти з англомовної Вікіпедії та Спрощеної англомовної Вікіпедії (Simple English Wikipedia), у той час як для інших мов такі навчальні дані ще збираються [16]. Далі з корпусу текстів потрібно вручну розробити стандартизовані правила, які ляжуть в основу майбутньої програми. Зразком можуть слугувати розміщені на сайті Inclusion Europe правила, при дотриманні яких текст має бути простим для людей з низькорозвиненими навичками читання. На ці правила можна орієнтуватися при написанні програми з автоматичної симпліфікації [31]. Якщо система створюється за допомогою технік машинного або глибокого навчання, то правила симпліфікації або пишуться розробниками вручну, або модель самостійно відшукує зв'язки між реченнями з оригінальних та спрощених текстів і таким чином навчається здійснювати симпліфікацію. Враховуючи різні способи створення систем машинного та глибокого навчання, це питання буде розглянуто більш детально в наступному розділі.

Існує три основні підходи до розробки правил для програм зі спрощення текстів. Розробляти правила можна вручну (Chandrasekar et al., 1996; Siddharthan, 2002; Siddharthan, 2010; Siddharthan, 2011 (усі для англійської мови)), автоматично з використанням алгоритмів машинного навчання (Woodsend and Lapata (2011), Zhu et al. (2010), Narayan and Gardent (2014) (усі для англійської мови)) або поєднувати два підходи (Brouwers et al. (2014) для французької мови; Bott and Saggion (2014) для іспанської мови; Caseli et al. (2009) і Specia (2010) для португальської мови у Бразилії; Aranzabe et al. (2013) для баскської мови [9], CLS (2021) для китайської мови [25]).

При першому підході увага лінгвістів спрямована на усунення ознак складности тексту. В основному це стосується синтаксичного рівня, а саме складних речень, відокремлених членів речення тощо (залежно від того, про спрощення текстів якою мовою йдеться). Другий підхід став можливим завдяки появі оригінальної та спрощеної версій одних і тих самих текстів (Англійська вікіпедія (English Wikipedia (EW)) та Спрощена англійська вікіпедія (Simple English Wikipedia (SEW))), а також вирівнюванню цих текстів за реченнями. Вирівнювання текстів за реченнями означає, що речення з EW та SEW мали бути розміщені в читабельному для програми вигляді (здебільшого в табличному), де одному реченню з EW відповідає результат спрощення цього речення (тобто уривок із SEW) [9]. Вирівняні речення є навчальними даними.

Розробка правил вручну та автоматично має свої переваги та недоліки. Перший підхід дозволяє розробникам контролювати зміни, які будуть відбуватися зі складними текстами при їх обробці програмою, але, по-перше, написання правил вручну вимагає фінансових витрат з боку замовника, а, по-друге, оскільки неможливо передбачити всі варіанти речень, написані вручну правила не завжди можуть спростити всі складні

синтаксичні та лексичні одиниці. У такому разі автоматична розробка правил на основі навчальних даних може бути корисною, але її ефективність напряму залежить від кількости навчальних даних, тому цей підхід також не може бути остаточним розв'язком завдання [розробка правил] [9].

Використання обох підходів при створенні правил для програми з автоматичної симпліфікації можна вважати оптимальним рішенням. Варіанти поєднання написаних вручну та створених автоматично правил можуть бути різними, що підтверджують приклади роботи деяких систем. Приміром, система Siddharthan and Angrosh (2014) утворена завдяки поєднанню підходів: вона функціонує на основі автоматично розроблених лексичних правил і прописаних вручну синтаксичних правил. Прикладами програм, симпліфікація яких базується на монолінгвальному паралельному корпусі, що розмічений правилами, які мають перетворити складні та ускладнені речення на більш прості, є Brouwers et al. (2014) (французька мова), Bott and Saggion (2014) (іспанська мова), Caseli et al. (2009) (португальська мова у Бразилії). Програма Specia (2010) (португальська мова у Бразилії) відрізняється тим, що її підхід до спрощення базується на машинному навчанні, утвореному за допомогою виведених з паралельного корпусу правил, заснованих на фразах. Aranzabe et al. (2013) (баскська мова) використовує результат програми, яка визначає читабельність кожного конкретного речення, а потім, якщо програма вирішила, що вони складні для сприйняття, перетворює їх за допомогою розроблених вручну правил [9].

Оскільки саме складний синтаксис викликає найбільше труднощів у розумінні текстів, то незалежно від підходу в розробці правил, процес симпліфікації найчастіше починається з пошуку програмою меж між реченнями, предикативними частинами складних речень, відокремленими

та другорядними членами речень тощо. Цей крок можна реалізувати завдяки синтаксичному парсеру (syntactic parser). Якщо на цьому етапі програма припускається помилок, далі процес перетворення дає незадовільний результат. Так, відповідно до Drndarevi´c et al. (2013), третина помилок при симпліфікації трапляється через попередні помилки синтаксичного парсеру. Brouwers et al. (2014) виявили, що 89% помилок спрощення є результатом попередніх помилок [9].

Підсумовуючи, можна сказати, що симпліфікація є однією з найзатребуваніших операцій у сучасній прикладній лінгвістиці. Вона дозволяє зробити зміст складного тексту доступнішим для людей з певними хворобами, неносіїв певної мови, дітей. Завдяки симпліфікації можливий подальший розв’язок інших мовних задач, наприклад здобування інформації про семантичні зв’язки (Relation Extraction), розмічування семантичних ролей (Semantic Role Labeling), машинний переклад (Machine Translation) тощо. Спрощення здійснюється вручну або автоматично на основі розроблених вручну та/або автоматично правил. Написані розробниками та сформульовані програмою в процесі її навчання на спрощених текстах правила мають свої переваги та недоліки. Поєднання цих двох підходів може бути реалізованим по-різному, але одночасне застосування їх є оптимальним при написанні програми. Більшість програм починають спрощення з визначення меж синтаксичних конструкцій за допомогою синтаксичного парсеру, але оскільки його робота може давати помилкові результати, які призводять до помилок на наступних етапах спрощення, покращення ефективности функціонування програм починається з корегування роботи парсеру. На сьогодні ще немає загальнодоступної програми симпліфікації для української мови. Проте ми маємо досвід інших мов, на який можна спиратися при створенні своєї системи.

1.2. Сучасні методи синтаксичної симпліфікації

Симпліфікація тексту може здійснюватися різними способами. Здебільшого вона починається із синтаксичного аналізу кожного речення та подальшого усунення або перетворення тих елементів, які можуть ускладнити сприйняття змісту. Ідеться про складні речення, відокремлені члени речення, вставні конструкції з додатковою інформацією, інверсію тощо.

Найбільший розвиток симпліфікації спостерігається в англійській мові. Хоч її граматика відрізняється від української, деякі підходи можуть бути корисними для створення програми автоматичного спрощення текстів нашою мовою.

Уперше синтаксичну симпліфікацію запропонували Chandrasekar et al. [1996]. Вони розробили систему, засновану на правилах, яка здійснює спрощення за допомогою синтаксичного аналізатора. Здебільшого їхньою метою було перетворення складнопідрядних з'ясувальних та означальних речень на два прості і поділ речення з прикладкою на два прості. Їхня робота дала поштовх до розвитку сучасних підходів у симпліфікації [31].

Прикладом одного з їхніх правил може слугувати формула

$$X: NP, RelPron\ Y, Z \rightarrow X: NP\ Z.\ X: NP\ Y.\ ,$$

де у складнопідрядному реченні головна частина починається на т. зв. «іменникову фразу» (noun phrase – NP), що позначається X, і має продовження, яке має позначку Z. Ця предикативна частина розривається підрядною, що починається на «відносний займенник» (relative pronoun – RelPron) та має продовження, позначене Y. У результаті симпліфікації

одержуємо 2 прості речення: обидва починаються на X, але в першому після X відразу йде Z, а у другому після X відразу йде Y [11].

(1) Talwinder Singh (X), who masterminded the Kanishka crash in 1984 (Y), was killed in a fierce two-hour encounter... (Z)

(1) Телвіндер Син (X), який спланував катастрофу під Корком 1984-ого року (Y), був убитий у результаті жорстокої двогодинної сутички… (Z)

↓

(2) Talwindcr Singh (X) was killed in a fierce two-hour encounter... (Z) Talwinder Singh (X) masterminded the Kanishka crash in 1984 (Y).

(2) Телвіндер Син (X) був убитий у результаті жорстокої двогодинної сутички… (Z) Телвіндер Син (X) спланував катастрофу під Корком 1984-ого року (Y). [11]

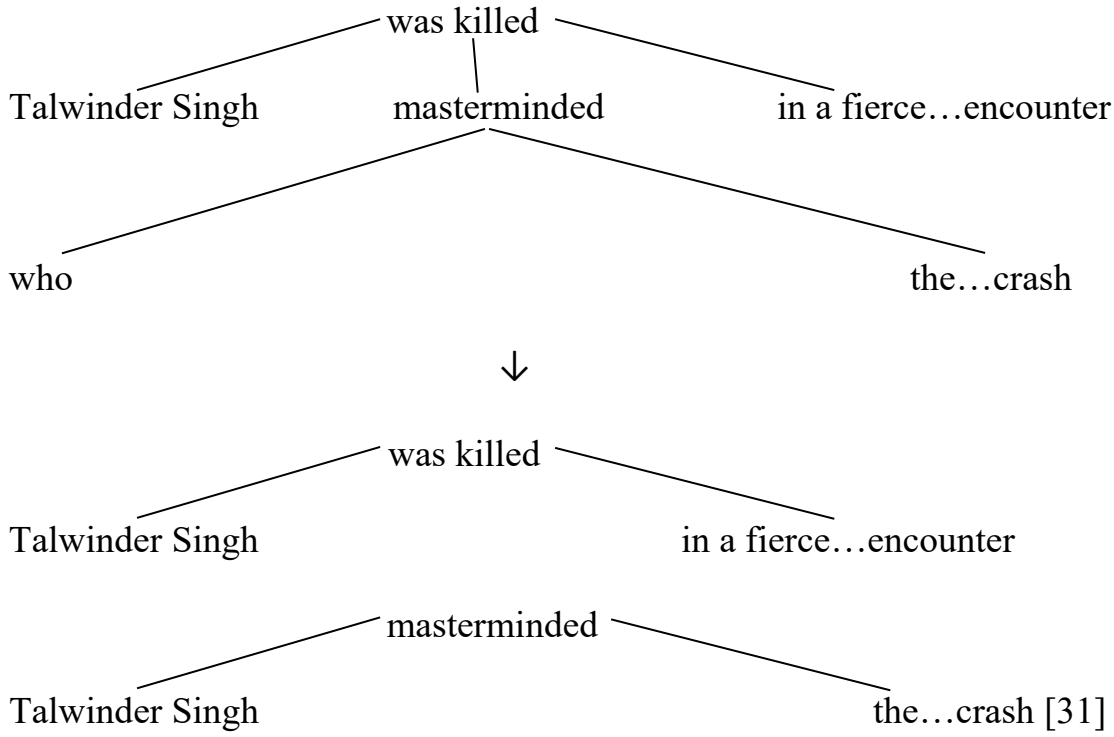
При цьому сполучний засіб у підрядній предикативній частині обов’язково має виконувати функцію підмета, щоб під час симпліфікації його можна було замінити на те слово з головної частини, якого стосується підрядна.

Робота системи (Chandrasekar, 1994), яка містить опис наведеного правила, була протестована на новинних текстах обсягом 224 речення. Результатом симпліфікації згідно з правилом стали 369 речень [11].

Згодом Chandrasekar and Srinivas [1997] запропонували прискорити розробку правил шляхом машинного навчання, тобто [шляхом] програмного формулювання правил на основі паралельного корпусу [31]. Було здійснено парсинг оригінальних і спрощених речень за допомогою Lightweight Dependency Analyzer [39]. Вхідні і вихідні речення порівнюються з алгоритмом, що допомогає визначити, які трансформації потрібні для перетворення вхідного дерева у вихідне дерево. Трансформації

включають змінні, які створюються за допомогою механізму задоволення обмежень (constraint satisfaction mechanism). Потім правила узагальнюються, змінюючи конкретні слова на теги. Для того, щоб навчити програму функціонувати разом з виробленими автоматично правилами, було застосовано тренувальний сет обсягом 65 текстів [31].

Можна проілюструвати одне зі створених описаним вище алгоритмом правил на прикладі вже згаданого речення. Але перед цим слід підкреслити, що хоч розробники системи Chandrasekar and Srinivas [1997] і мали на меті підвищити ефективність симпліфікації за меншу кількість часу, вони все одно потребували часу на те, щоб зібрати достатню кількість навчальних даних [31].



На перший погляд ідея здається простою. Однак автори Siddharthan [2006] помітили, що при наведеній вище процедурі значення тексту може спотворитися [31]. Спробуймо спростити речення за таким самим принципом.

(1) Mr. Anthony, who runs an employment agency, decries program trading, but he isn’t sure it should be strictly regulated.

(1) Містер Ентоні, що керує агентством з працевлаштування, критикує програмну торгівлю, але він не впевнений, що це має бути суворо регульованим.

↓

(2) Mr. Anthony decries program trading. Mr. Anthony runs an employment agency. But he isn’t sure it should be strictly regulated.

(2) Містер Ентоні критикує програмну торгівлю. Містер Ентоні керує агентством з працевлаштування. Але він не впевнений, що це має бути суворо регульованим [31] [37].

Для уникнення таких розбіжностей у змісті оригінального та спрощеного текстів було розроблено архітектурну систему з трьома етапами: аналіз, трансформація та регенерація. Система містить у собі вісім правил, які стосуються різних граматичних конструкцій англійської мови (Adjectival (or Relative) Clauses; Adverbial clauses; Coordinated Clauses; Subordinated clauses; Correlated clauses; Participial Phrases; Appositive Phrases; Voice). З переліку видно, що не всі з конструкцій мають «відповідники» в українській мові. Система влаштована таким чином, що до одного речення можна застосовувати тільки одне правило на етапі трансформації. Регенерація відповідає за т. зв. «сполучувальну єдність» (conjunctive cohesion) і «анафоричну єдність» (anaphoric cohesion) [35]

Серед підходів у синтаксичній симпліфікації слід виокремити метод ключових подій. Він пов’язаний із семантичним аналізом речення і складається з 2-ох компонентів: перший має визначити слова, які містять у собі основний зміст описаних у реченні подій, а другий, базуючись на сформульованих розробниками правилах, реалізує спрощення. Після

знаходження слів-носіїв основного змісту їм присвоюється один з 4-ох тегів: фактор (agent), мета (target), час (time) або місце (location) [31]. Розглянемо застосування цього методу на прикладі.

China confronted the Philippines. – nsubj (confront, China) – agent

(Китай зіштовхнувся з Філіппінами.) [31]

У наведеному прикладі система визначила слово *confront* як носія основного змісту, поєднаного зі словом *China* предикативними відношеннями. Завдяки цьому можливо присвоїти *China* аргумент *agent* [31].

Після знаходження слів-носіїв змісту та присвоєння їм аргументів є 2 способи подальшого спрощення: sentence-wise метод, коли слова та фрази, не помічені програмою і, відповідно, не мають одного з тегів *фактор*, *мета*, *час* або *місце*, визначаються як зайві та видаляються з тексту, та event-wise метод, коли кожна помічена подія бере участь у створенні спрощеного варіанту тексту. При цьому в останньому методі події, що містять інформацію звітного характеру (reporting events), та події, помічені за допомогою слів-носіїв, ігноруються, тому що вони [події, помічені за допомогою слів-носіїв], ймовірно, мають кілька тегів. Також в event-wise методі виражена за допомогою герундія подія, що керує подією головного речення, перетворюється на речення в минулому часі. Після цього програма копіює вхідні слова у вихідні речення, обмежуючись мінімальними перетвореннями [31]. Розгляньмо реалізацію описаних кроків на прикладі.

Оригінал: Baset al-Megrahi, the Lybian intelligence officer who was convicted in the 1988 Lockerbie bombing has died at his home in Tripoli, nearly three years after he was released from a Scottish prison.

Бесет ол-Меґрагі, лівійський офіцер розвідки, що був засуджений 1988-ого за вибух над Локербі, помер у себе вдома у Тріполі, приблизно після трьох років він був випущений із шотландської в'язниці.

Sentence-wise метод: Baset al-Megrahi was convicted in the 1988 Lockerbie bombing has died at his home after he was released from a Scottish prison.

Бесет ол-Меґрагі був засуджений 1988-ого за вибух над Локербі, помер у себе вдома, після він був випущений із шотландської в'язниці.

Event-wise метод: Baset al-Megrahi was convicted in the 1988 Lockerbie bombing. Baset al-Megrahi has died at his home. Baset al-Megrahi was released from a Scottish prison.

Бесет ол-Меґрагі був засуджений 1988-ого за вибух над Локербі. Бесет ол-Меґрагі помер у себе вдома. Бесет ол-Меґрагі був випущений із шотландської в'язниці [31].

Яскравим прикладом сучасного застосування цього методу є система ERNESTA, розроблена для італійської мови. Ця система спрощує тексти для 7-11-річних дітей з низькими навичками читання. Її завданнями є пошук ключових подій, про які йдеться в тексті, видалення з тексту необов'язкових елементів і перетворення здобутої інформації на прості речення в теперішньому часі [2].

Можна зробити висновок, що синтаксична симпліфікація є основою спрощення текстів. Вона має на меті перетворити складні речення з кількома предикативними частинами, відокремленими членами речення тощо на прості і при цьому зберегти основний зміст тексту. На сьогодні існують методи для здійснення синтаксичної симпліфікації. Здебільшого вони стосуються англійської мови. Інформація про сучасні підходи та ідеї реалізації спрощення синтаксису може бути корисна і для української.

Правила спрощення можна частково адаптувати до нашої мови, адже у граматиках англійської та української наявні спільні риси, але все одно потрібно заглиблюватись у наш синтаксис, відслідковувати аналогії, щоб у подальшому на їхній основі розробити власні правила синтаксичної симпліфікації (якщо ідеться саме про ручну розробку правил).

1.3. Актуальні методи лексичної симпліфікації

Лексична симпліфікація – спрощення лексичного рівня тексту. Її метою є заміна складних або маловживаних слів на легші для розуміння відповідники разом зі збереженням основного змісту тексту. З поданих у цьому підрозділі прикладів роботи деяких систем видно, що спершу має здійснюватися синтаксична симпліфікація і тільки після неї лексична. Якщо, наприклад, є складне речення з кількома предикативними частинами, то в результаті лексичного спрощення буде здійснено заміну слів, визначених програмою як складних, але речення не стане легшим для сприйняття. У такій ситуації спершу потрібно перетворити складне речення на кілька простих, а вже кожне з них піддавати лексичній симпліфікації.

Для досягнення оптимального результату лексичного спрощення потрібно, по-перше, знайти список синонімів, які можуть замінити нечитабельне слово, по-друге, замінити його на такий синонім, який буде найбільш доречним у певному контексті [31]. Для вирішення першого завдання можна використовувати лексичні ресурси, наприклад, WordNet [5]. Виконання другого завдання можливе при застосуванні алгоритмів зняття омонімії (word sense disambiguation – WSD) та вирахування рівня простоти тексту. При цьому системи спираються: 1) або на частоту слова як на показник простоти (Lal and Rüger, 2002) [24], 2) або на параметр довжини

слова, який свідчить про його складність або простоту (Bautista et al., 2011) [3], 3) або поєднуються обидва згадані підходи (Bott et al., 2012) [8].

Carroll et al. [1998] були першими, хто запропонував систему лексичної симпліфікації для англійської мови. Кожне слово у вхідному тексті перевіряється у WordNet. До кожного слова створюються списки синонімів за допомогою бази у WordNet. Далі Оксфордська психолінгвістична база даних (Oxford Psycholinguistic Database) має визначити частоту кожного слова та замінити рідкісні на частотніші синоніми. Наприклад, у реченні “My automobile is broken.” («Мій автомобіль зламаний») слово *automobile* має синоніми *car, auto, machine* та *motorcar* з частотами 278, 26, 108, 0 відповідно. Частота *automobile* дорівнює 50. Отже, у результаті симпліфікації маємо: “My car is broken.” («Мій автомобіль зламаний») [31].

Цей підхід був детальніше вивчений Shardlow [2014], чиї розробники шукали способи покращення його роботи шляхом пошуку можливих джерел помилок. Під час тестування своєї ж системи вони використали 115 новинних статей. Протягом перевірки на першому етапі роботи програми (ідентифікація складних слів) було здійснено 183 лексичні заміни, 119 з яких виявилися помилковими. Наступний крок (заміна складних слів на прості) було зроблено із 64-ма словами (183-119=64), 27 з яких [замін складних слів на прості] також виявилися помилковими. У підсумку система припустилася 7-ми видів помилок різного характеру (99 помилок зі 164 полягали в присвоєнні статусу “складних” тим словам, які складними не були) і лише 19 замін зі 183 виявилися повністю правильними [34].

Ще одна система, варта уваги, це LexSiS. Вона здійснює лексичну симпліфікацію текстів іспанською мовою. Її підхід схожий на Carroll et al.. При цьому вона враховує і довжину, і частоту слова. LexSIS базується на Іспанському відкритому словнику (Spanish Open Tesaurus – SOT), який

нараховує 21,831 слів (лем) і подає списки зі словами, які можна поставити на місце певної леми. Спрощення відбувається таким чином: для леми, яка визначається програмою як складне слово, створюється вектор слова (word vector) з урахуванням контексту. Цей вектор порівнюється з усіма можливими векторами леми у словнику, щоб вибрати найближчий вектор. Далі зі списку синонімів, який містить і вихідне слово, вибирається найкращий варіант для заміни [8].

Для визначення складности слова використовується векторна словесна просторова модель (word vector space model). У ній значення слова подається в контексті, де це слово може бути виявленим. Значення слів подаються в LexSIS як вектори. Вектори всіх синонімів для певного слова застосовуються для утворення т. зв. «смислового вектора» (sense vector). Векторна модель слова має бути виведена з великого обсягу даних. Для LexSIS це було зроблено на основі корпусу іспанських новин, що складається з 8 мільйонів слововживань. Коли в тексті помічена омонімія, застосовується 2 методи для вибору найдоречнішого слова для заміни. Локальний метод розглядає мінімум контексту, шукаючи інформацію, достатню для зняття омонімії. Глобальний метод розглядає всі випадки, де слово, яке розглядається, трапляється в тексті [31]. Потім подаються всі контексти та висувається припущення про один сенс на дискурс (односторонній дискурс) [13]. У будь-якому разі вектори міститимуть леми та будуть скоріше рідкісними [31].

LexSIS не замінює слово, яке привернуло увагу програми, якщо воно вже має високу частоту або коли вибрана заміна надто відрізняється від слова, яке розглядається (це визначається за допомогою схожости векторів першого і другого слів) [31].

Ця система розвинулася завдяки опитуванню трьох іспанських мовців. В анкеті були пари речень, які відрізнялись одним словом. Заміна

певних слів здійснювалася за допомогою різних методів, зокрема підходу, який є в LexSIS. Респонденти не знали, яке речення є оригіналом, а яке – результатом спрощення. Для кожної пари речень потрібно було схарактеризувати своє сприйняття з точки зору складности, послуговуючись відповідями «однакова складність», «більш складне», «мають різні значення», «не є зрозумілими». Виявилося, що LexSIS здійснює більше синонімічних замін (72%), ніж частотний метод (66%), схожий на Carroll et al. [1998] та описаний вище. Здебільшого дібрані LexSIS синоніми були простіші або мали однакову складність з вихідними словами [31].

Для коректної лексичної симпліфікації потрібно визначити складність слова. Завдання з визначення складних слів (Complex Word Identification task) було запропоноване саме для цього. Воно здійснювалося на основі корпусу складних слів (CW corpus) [33], LexMturk corpus [19] і Спрощеної англійської Вікіпедії (Simple English Wikipedia) [23]. 20-м респондентам, які не є носіями англійської, було запропоновано виокремити іменники, дієслова, прикметники та прислівники у 200 реченнях. У процесі визначення названих частин мови опитані мали зазначати, яке слово з іменників, дієслів, прикметників та прислівників є незрозумілим для них, навіть якщо вони розуміли все речення загалом. 9000 речень були так само анотовані, але однією людиною, яка теж не є носієм англійської. У результаті було зібрано 2 бази даних: оптимістична – зі словами, які були визначені як складні хоча б одним респондентом, та консервативна – зі словами, які були визначені як складні хоча б п’ятьма опитаними [31].

Після цього 20 команд представили свої системи з виявлення складних слів. Вони були оцінені за такими параметрами: точність, чіткість (precision), відгук (recall), точність, достовірність (accuracy), F бали (F-score)

та G бали (G-score). Останні 2 параметри означають гармонійне середнє значення між recall та accuracy. З точки зору G-score найкращими командами виявилися the organizers [Paetzold and Specia, 2016b] і the TALN team [Ronzano et al., 2016], а з точки зору F-score – Wrobel [2016] та Malmasi et al. [2016] [31].

Правила лексичної симпліфікації в системі Biran et al. [2011] також базуються на текстах EW та SEW. Вона використовує контекстні вектори, щоб знайти пари слів, які трапляються в однакових контекстах в EW та SEW. WordNet слугує фільтром для можливих лексичних змін. Ця система визначає складність слів, ураховуючи і частоту, і довжину. Береться пара синонімів, потім частота слова з EW ділиться на частоту слова з SEW. Далі одержане число треба помножити на довжину кожного слова. Таким чином вийде одержати складність і першого, і другого. Наприклад, у парі слів (canine, dog (собака, собака)) canine завжди можна замінити на dog, але не навпаки, оскільки складність першого є вищою, ніж у другого. При цьому під час симпліфікації у Biran et al. [2011] діє метод, який досліджує контекст і допомагає визначити, яке слово найкраще підходить у певній ситуації, фільтрує правила, що спотворюють зміст (context-aware method) [7]. Цей підхід нагадує метод в LexSIS, тільки там застосовується vector space model.

При лексичній симпліфікації розробники можуть вдатися до застосування лінгвістичних моделей. Наприклад, De Belder et al. [2010] використовує 2 джерела інформації: словник синонімів і лінгвістичну модель прихованих слів (latent words language model – LWLM). LWLM на основі великого за обсягом корпусу навчається добирати до кожного слова список синонімів і слів, які можуть підійти для заміни. Вона здійснює зняття омонімії (word sense disambiguation – WSD) або відкидає ті слова, які визначить як зайві. Слова, запропоновані LWLM, можуть не враховуватися, якщо їх немає в авторитетному списку синонімів. Саме так функціонує

ймовірнісний підхід, спираючись на 2 джерела інформації – інформацію з лінгвістичної моделі та ймовірність, що синонім є легким для сприйняття. Розробники стверджують, що останнє можна визначити за допомогою психолінгвістичної бази даних, позначивши частоти як значення на проміжку [0, 1] [31].

Окрім цього, існує метод лексичної симпліфікації, що базується на дистрибуційному лексико-семантичному підході. У своїй роботі його використовує система Glavaš and Štajner [2015]. Вона базується на EW та на корпусі Gigaword 5 corpora і моделює семантику слова за допомогою векторів слів з GloVe – сучасного інструмента лексичного моделювання. Свій підхід розробники називають LIGHT-LS. Якщо є слово, яке потрібно замінити, програма дістає такі слова з вкладеної словесної моделі, чиї вектори найближчі до складного слова. Далі дібрані слова класифікуються за такими критеріями: семантична схожість зі складним словом, контекстна схожість зі складним словом, порівняння інформації зі слова-кандидата для заміни та складного слова і ступінь придатности слова-кандидата в контексті (вимірюється з імовірністю 5-грамної лінгвістичної моделі) [17].

Окремо варто сказати про спрощення числових виразів у текстах. Яскравим прикладом орієнтованої на перетворення числових виразів системи є іспанськомовна Bautista and Saggion [2014] [4]. Вона використовує аналізатор для поділу речень, маркування, присвоєння словам частиномовних тегів, далі за допомогою набору граматичних правил визначає числові вирази, після цього здійснює симпліфікацію завдяки розробленим правилам і повертає перетворений текст. Складені числові вирази мають тег Zd (наприклад, *16,4 мільйонів*). Для перетворень застосовується 45 граматичних правил, розроблених унаслідок роботи з корпусом та опитування іспаномовних людей. Наприклад, одне з правил

трансформації числових виразів виникло саме після анкетування. Ідеться про зміну на зразок *3,9 мільйонів людей* на *майже 4 мільйони людей* [31].

Отже, лексична симпліфікація замінює складні (багатоскладові і рідко вживані) слова на їхні синоніми для полегшення сприйняття тексту читачами. На сьогодні досягти цієї мети можна за допомогою багатьох методів. У підрозділі подано лише частину з них. Англійська є однією з найрозвиненіших у цьому плані мов: саме для неї розроблено чи не найбільше підходів, на неї орієнтуються розробники програм для інших мов.

РОЗДІЛ 2

РЕАЛІЗАЦІЯ СИМПЛІФІКАЦІЇ ІЗ ЗАСТОСУВАННЯМ ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

2.1. Історія використання штучних нейронних мереж в обробці природної мови

Симпліфікація текстів разом з машинним перекладом, здобуванням інформації про семантичні зв’язки, сентимент-аналізом та іншими завданнями належить до такої галузі, як обробка природної мови (Natural Language Processing (NLP)). Зріст обчислювальної потужности комп’ютерів і розпаралелювання, спричинений завдяки графемній обробці одиниці

(Graphical Processing Units (GPUs)), дозволяє покращувати результати вищезгаданих та інших завдань прикладної лінгвістики шляхом застосування алгоритмів глибокого навчання (deep learning) [21]. Це один із найсучасніших способів реалізації задач NLP, зокрема симпліфікації [14], про історію розвитку якого йтиметься в цьому розділі.

Робота алгоритмів глибокого навчання передбачає створення та використання штучних нейронних мереж (ШНН). Штучна нейронна мережа (ШНМ, artificial neural networks (ANNs)) – це система, яка імітує роботу людського мозку і складається з нейронів, об'єднаних у шари. Один нейрон, що має зв'язки з іншими нейронами з попереднього шару (або з усіма з попереднього шару), може отримувати сигнали від нейронів попереднього шару і передавати їх на наступний [18]. Із самої назви *глибоке навчання*, а також із визначення, де згадано про імітацію роботи людського мозку, стає зрозумілим, що ШНМ влаштована таким чином, що навчається виконувати певні функції на тренувальному сеті даних і, власне, виконує їх. Таким чином, ШНМ пов'язані з поняттям штучного інтелекту (ШІ), тому, окрім історії появи та розвитку самих ШНМ для реалізації завдань NLP, доцільним є окреслити найважливіші поняття, пов'язані із ШІ.

ШІ має 2 концепції, а саме сильний ШІ (strong AI) та слабкий ШІ (weak AI). Коли було введено термін *ШІ*, він означав систему, яка функціонує так само, як людський мозок (зараз це є дефініцією сильного ШІ). Ідея сильного ШІ бере свій початок у визначенні інтелекту як "здатности системи функціонувати належним чином в умовах невизначености". Відповідно до цієї дефініції, людський інтелект хоч і не є досконалим, але має здатність впоратися із ситуаціями, що виникають у найпоширеніших умовах невизначености. А слабкий ШІ означає когнітивну та критичну систему, яка притаманна обчислювальній техніці і не має на меті відтворити людський інтелект, як це очікується від сильного ШІ.

Слабкий ШІ призначений для виконання інтелектуальної діяльности, з якою може впоратися людина. Ця ідея виникла, коли стало зрозуміло, що, по-перше, реалізація обчислень істотно відрізняється від інтелекту людини і, по-друге, всебічна людська особистість не є обов'язковою для ефективних обчислювальних операцій [29].

Машина Тюрінга є основою сучасної ідеї ШІ. У 1950 р. А. Тюрінг представив гру в імітацію (imitation game), також відому як Тест Тюрінга (Turing Test). Цей тест мав вважатися пройденим, якщо людина не може відрізнити іншу людину від ШІ під час діалогу з нею [40]. З появою чату GPT можна стверджувати, що ця система [GPT] пройшла Тест Тюрінга [20]. Знаючи різницю між сильним та слабким ШІ, зрозуміло, що А. Тюрінг під ШІ мав на увазі саме сильний ШІ.

Повертаючись до ШНМ в NLP, початок їх використання в цій галузі датується 2001 роком, коли ШНМ було вперше запропоновано для передбачення наступного слова в тексті на основі попередніх слів. Ідеться про модель прямого поширення (feed-forward model) від Bengio et al. [Додаток 1]. Ця модель приймає представлення попередніх слів, які знаходяться в table C, як вхідний вектор. Зараз такі слова називаються вбудованими (word embeddings). Вбудовані слова об'єднуються і передаються у прихований шар, а вивід прихованого шару переходить у шар softmax. За деякий час для здійснення мовного моделювання моделі прямого поширення було замінено на повторювані нейронні мережі (recurrent neural networks, RNN; Mikolov et al., 2010) та мережі довготривалої короткочасної пам'яти (long short-term memory networks, LSTM; Graves, 2013; Melis et al., 2018). Разом з тим класична нейронна мережа прямого поширення Bengio et al. все ще лишається конкурентноспроможною порівняно з більш складними, оскільки останні при передбаченні наступних слів у тексті зазвичай вчаться враховувати

лише нові слова (Daniluk et al., 2017). Мовне моделювання є формою неконтрольованого навчання (unsupervised learning) [30].

Зріст ефективности мовного моделювання відбувся у 2013 р. завдяки Mikolov et al., чиї розробники запропонували покращити тренування вбудованих слів, прибравши прихований шар і наблизивши в такий спосіб шар softmax. Також вони представили алгоритм, що використовує мережу для вивчення асоціацій між словами на основі корпусу тексту (цей алгоритм відомий під назвою word2vec). Ці кроки виправдали себе: вони зробили можливим більш повне навчання вбудованих слів. Архітектура системи представлена у двох варіантах: continuous bag-of-words (CBOW) (мета – передбачити центральне слово, враховуючи слова, які його [ще не передбачене центральне слово] оточують) та skip-gram (мета повністю протилежна) [Додаток 2]. Хоч вкладення слів не відрізняються принципово від мережі прямого поширення, тренування їх [вкладень слів] на основі великого за обсягом корпусу дозволило їм сформувати зв’язки між такими словами, як стать, форма дієслова, країна-столиця [Додаток 3] [30].

Приблизно в той самий час ШНМ почали широко застосовуватися в NLP. Набільш поширеними типами ШНМ стали згорткові (Kim, 2014 [Додаток 4]) та рекурсивні (Socher et al., 2013 [Додаток 5]) нейронні мережі. Згорткові ШНМ, створені для роботи з текстовими даними, працюють за допомогою двох вимірів з фільтрами, які потрібно переміщувати вздовж тимчасового виміру. Рекурсивні ШНМ є втіленням ідеї про представлення речення як дерева, а не як послідовности: значення кожного вузла обчислюється за допомогою складання значень вузлів з попереднього(-іх) рівня(-ів) [30]. Класифікацію ШНМ та пояснення принципів роботи кожного їх виду більш детально описано в наступному підрозділі.

У 2014 р. творцями системи Sutskever et al. було запропоновано навчати ШНМ за принципом перетворення послідовности на послідовність

(sequence-to-sequence model). У структурі таких ШНМ мережа енкодера обробляє кожен символ вхідного речення і стискає це речення у векторне представлення, а мережа декодера передбачає кожен символ вихідного речення, орієнтуючись на зроблене енкодером векторне представлення та враховуючи передбачений під час попереднього кроку символ [Додаток 6]. Ця ідея стала поштовхом для розвитку машинного перекладу: у 2016 р. Гугл оголосив про заміну моделей, що навчаються на фразах (phrase-based models) на нейронні мережі, що скоротило програмний код з 500 тис. рядків до 500. Модель, що працює за принципом перетворення послідовности на послідовність і виконує функції енкодера та декодера, дозволяє реалізовувати завдання NLP, які передбачають генерування результату. Але оскільки робота декодера може бути зумовлена не тільки послідовністю вхідних символів, а й довільними представленнями (тобто декодер може аналізувати й усі символи разом, а не лише кожен окремо взятий символ), завдяки ідеї перетворення послідовности на послідовність стали можливими такі операції, як генерування текстових представлень інформації з таблиць (Lebret et al., 2016), опису змін початкового коду (Loyola et al., 2017) тощо [30].

Підхід під назвою *увага* (attention), представлений авторами Bahdanau et al. (2015), зміг ще більше полешити виконання машинного перекладу. На відміну від описаного вище принципу перетворення послідовности на послідовність, увага дозволяє декодеру “озиратися” на приховані стани вхідної послідовности. Потім приховані стани вхідної послідовности об’єднуються в одне вхідне для декодера значення [Додаток 7]. Цей спосіб найбільше підійде тим завданням, результатом яких має бути рішення, ухвалене на основі деяких частин вхідних даних, зокрема для парсингу складових (constituency parsing) (Vinyals et al., 2015), розуміння

прочитаного (Hermann et al., 2015) та одноразове навчання (one-shot learning) (Vinyals et al., 2016) [30].

Одним з найновіших підходів у побудові ШНМ є попередньо навчені мовні моделі. Для підготовки таких моделей потрібні тільки нерозмічені тексти, тому масштаб навчання може розширитися до мільярдів токенів, нових доменів і нових мов. Хоча застосування попередньо навчених моделей було вперше запропоновано у 2015 р. (Dai & Le, 2015), свою ефективність при реалізації різних завдань вони проявили приблизно у 2018 р. . Вкладення моделей можна використати як ознаки в цільовій моделі (Peters et al., 2018), а саму модель – точно налаштувати на даних цільового завдання (Ramachandran et al., 2017; Howard & Ruder, 2018). Попередньо навчені мовні моделі показали здатність до навчання з високою ефективністю на значно меншій кількості даних. Оскільки мовні моделі потребують лише нерозмічених даних, вони особливо корисні для мов з обмеженими ресурсами, де такі дані відсутні [30].

Застосування алгоритмів глибокого навчання допомагає покращити ефективність симпліфікації текстів як одного із завдань NLP. Глибоке навчання реалізовується шляхом створення ШНМ, яка імітує роботу людського мозку і пов’язана із поняттями ШІ (сильного та слабкого) і Тесту Тюрінга, який мав на меті реалізувати сильний ШІ (на зразок сучасного чату GPT). Уперше ШНМ застосовано в NLP у 2001 р. . Відтоді було створено кілька видів ШНМ, навчання яких дозволяє виконувати різноманітні завдання NLP. Можна орієнтуватися на вже готові підходи для розвитку NLP української мови, зокрема для втілення симпліфікації текстів.

2.2. Класифікація штучних нейронних мереж

Як зазначалось у попередньому підрозділі, нейронні мережі складаються із взаємопов'язаних, об'єднаних у шари зв'язків (нейронів), кожен з яких на вхід отримує дані з попереднього шару і після певної роботи з ними постачає їх на наступний шар. Вхід і вихід залежать від багатьох чинників: мети роботи мережі, її архітектури, обсягу баз даних, на яких вона навчається передавати сигнали, тощо.

Усі нейрони мають 2 кінці та по одному вузлу на кожному з кінців, тому нейронну мережу можна уявити як граф. Процес перетворення даних при передачі їх від шару до шару можна описати так: вузли у вихідному шарі отримують значення з вхідного шару, далі виконують обчислення зваженої суми цих значень і генерують вихідні дані за допомогою нелінійних функцій перетворення. При помилках або втраті якихось даних відбувається виправлення ваг значень, що можна побачити на вихідних вузлах. У сучасних мережах такі виправлення здійснюються з використанням стохастичного градієнта спуску, який називається зворотним поширенням [21].

Параметрами, за якими класифікують ШНМ, є способи з'єднання вузлів і кількість шарів. Поки немає чіткого визначення, яка мережа є глибокою, але є негласне правило, за яким мережі з множинними прихованими шарами вважають глибокими, а ті, що мають багато шарів, – дуже глибокими. Тобто теоретично можлива ситуація, коли окремо взята глибока мережа має більше шарів, ніж окремо взята дуже глибока. Основні мережі, у яких усі вузли організовано в послідовні шари і кожен вузол отримує вхідні дані лише від вузлів з попередніх шарів, називають мережами прямого зв'язку (feedforward neural networks (FFNNs)). Окрім цих, існує ще декілька типів мереж [32].

- Згорткові нейронні мережі (ЗНМ, Convolutional neural networks (CNNs))

ШНМ цього типу побудовані на неокогнітроні К. Фукусіми (неокогнітрон К. Фукусіми – це згорткова ієрархічна, багатошарова мережа, розроблена професором Куніхіко Фукусімою). Отримали свою назву від операції згортки в математиці та від обробки сигналів. ЗНМ використовують функції, відомі як фільтри, що дозволяє проводити одночасний аналіз різних ознак у даних. Ці мережі використовуються в обробці зображень, відео та мовлення. У роботі ЗНМ важливо, чи з’являються певні ознаки в окремих фрагментах мережі. Значить, операції об’єднання в пул можна використовувати для мінімізації розміру карт об’єктів (вихідні дані згорткових фільтрів). Але розміри таких пулів зазвичай замалі щоб зберегти високу точність [21].

- Рекурсивні нейронні мережі (РНМ, Recursive Neural Networks (RNN))

Як і ЗНМ, мережі цього типу використовують форму розподілу ваги для мінімізації навчання. Проте ЗНМ розподіляють ваги значень горизонтально (у межах шару), РНМ роблять це вертикально (між шарами). Вертикальний розподіл ваг значень дозволяє моделювати структури у вигляді дерев розбору. Також у мережах цього типу можна, починаючи з нижчого рівня дерев, використовувати єдиний тензор (алгебраїчний об'єкт, що описує багатолінійне відношення між множинами алгебраїчних об'єктів, пов'язаних з векторним простором (ідеться про такі об’єкти, як скаляр, вектор, ковектор, лінійний оператор і білінійна форма) або узагальнену матрицю ваг значень, а потім рекурсивно використовувати єдиний тензор або узагальнену матрицю на вищих рівнях [21].

- Повторювані нейронні мережі (ПНМ, Recurrent Neural Networks) та Мережі короткочасної пам’яти (МКП, Long Short-Term Memory Networks)

ПНМ є широко застосовуваним типом РНМ з попереднього пункту. Оскільки ефективність більшої частини завдань NLP залежить від порядку слів або інших елементів (фонеми, речення тощо), корисно мати змогу повернутися до інформації про попередні елементи при обробці нових. Іноді буває навпаки, коли правильна обробка слів залежить від наступних елементів. Тому корисно дивитися на речення в обох напрямках (уперед і назад), використовуючи 2 ПНМ-шари і поєднуючи їхні результати (називається двонаправленою ПНМ). Якщо в мережі цього типу існує послідовність шарів, це дає краще остаточне представлення результатів. Також ефект введення може залишатися довше, ніж під час роботи з одним ПНМ-шаром, що приводить до довгострокового ефекту. Комірки ПНМ називають стеком ПНМ [21].

Однією з найбільш високотехнологічних ПНМ є МКП. В МКП рекурсивні вузли складаються з кількох окремих нейронів, з'єднаних таким чином, щоб зберегти, забути або розкрити конкретну інформацію. Оскільки типові ПНМ з поодинокими нейронами, які технічно повертаються до самих себе, містять у пам'яті інформацію про давно минулі результати, ці результати "розбавляються" з кожною наступною ітерацією. Часто потрібно звернутися до давно минулої інформації, і завдяки використанню блоків МКП можна зберегти її набагато довше. Існує трохи простіший варіянт МКП, який називається *закритий повторюваний блок* (Gated Recurrent Unit (GRU)) і працює так само, а іноді навіть краще, ніж стандартна МКП [21] [6].

- Механізми уваги й Трансформер (Увага (Attention Mechanisms (AM) and Transformer))

Для таких завдань, як машинний переклад, анотування й реферування текстів або субтитрування, вихідні дані надходять у текстовій формі. Зазвичай це робиться за допомогою використання пар енкодер-декодер

(encoder–decoder) і працює таким чином: ШНМ для кодування виробляє вектор певної довжини, а ШНМ для декодування повертає змінну, значенням якої є довжина тексту, вирахувана на основі вектора. Але може виникнути така проблема: мережа змушена кодувати всю послідовність у вектор кінцевої довжини незалежно від того, чи є хоч які-небудь вхідні дані важливіші за інші вхідні дані [Додаток 8]. Надійним розв'язком є згаданий у попередньому підрозділі підхід Увага. Для реалізації цієї ідеї потрібно застосувати щільний шар для анотованого зважування прихованого стану мережі, що дозволяє їй [мережі] навчитися визначати, де приділяти увагу, відповідно до поточного прихованого стану та анотації [Додаток 9]. ШНМ з підходом Увага бувають згорткові (convolutional), внутрішньочасові (intra-temporal), закриті (gated) та самоуважні (self-attention). Останній передбачає приділення уваги словам в рамках одного речення. Наприклад, під час кодування слова у вхідному реченні корисно проєктувати змінну на позначення кількости уваги до інших слів у реченні. Під час декодування є сенс приділяти увагу словам, які вже були створені [21].

Самоуважність набула широкого застосування в найсучаснішій моделі енкодера-декодера під назвою Трансформер [Додаток 10]. Ця модель має низку накладених одне на одного енкодерів та декодерів, самоуважність у кожному блоці енкодера й декодера і перехресну увагу між енкодерами й декодерами. Вона одночасно використовує кілька екземплярів уваги, уникаючи повторів і згортань (convolutions). Трансформер є основним компонентом у багатьох найсучасніших нейронних мережах для NLP [21].

- Залишкові з'єднання та Відключення (Residual Connections (RC) and Dropout)

У натренованих за допомогою зворотного поширення глибоких мережах для виправлення частих помилок використовують градієнти. Цю

тенденцію можна зменшити вибравши таку функцію активації, як виправлений лінійний блок (Rectified Linear Unit (ReLU)), які не показують ареально кострубаті фрагменти або фрагменти з малими градієнтами. Для розв'язку цього та інших питань часто застосовують залишкові з'єднання. Ці з'єднання пропускають шари (звичайно один). Якщо з'єднання використати під час кожного чергування шару, це вдвічі скорочує кількість шарів, через які градієнт має поширюватися назад. Така мережа відома як залишкова мережа (ResNet). Окрім неї, існують ще мережі, включаючи Highway Networks і DenseNets [21].

Ще одним важливим методом для тренування ШНМ є відключення. При відключенні деякі з'єднання і, можливо, навіть вузли деактивуються випадковим чином для кожного навчального пакету (невеликий набір прикладів). Щоразу, у кожній порції навчального пакет відключення вибирає, які вузли деактивувати. Це змушує мережу розподіляти свою пам'ять кількома шляхами, допомагаючи в узагальненні та зменшуючи ймовірність переналаштування на навчальні дані [21].

Із наведеної класифікації видно, що ШНМ є різноманітними залежно від кількости шарів і способів з'єднання вузлів. Багатий перелік типів мереж (Мережі прямого зв'язку, Згорткові нейронні мережі, Рекурсивні нейронні мережі, Повторювані нейронні мережі, Мережі короткочасної пам'яти, Увага, Трансформер, Залишкові з'єднання та Відключення) дозволяє вибрати найбільш відповідний тип мережі і реалізовувати його для розв'язку кожного із завдань NLP, зокрема симпліфікації текстів.

2.3. Підходи реферування та генерування текстів при реалізації симпліфікації

Якщо симпліфікація тексту означає його синтаксичне та лексичне спрощення з одночасним збереженням найголовнішої інформації, то, залежно від вибраного способу реалізації, ця операція поєднує в собі ознаки текстового реферування і текстового генерування. Це зумовило існування двох підходів до симпліфікації, стислий опис яких подано в цьому підрозділі.

1. Екстрактивний підхід (Extractive Approach)

Екстрактивний підхід передбачає здобування речень з текстового фрагмента, вибираючи їх таким чином, щоб вони містили найважливішу інформацію, при цьому не змінюючи їх. Цей спосіб тяжіє до реферування текстів [26].

Найпростішою технікою текстового реферування є Частота слова – Інверсивна частота документа (Term Frequency – Inverse Document Frequency (TF-IDF)). Вона генерує таблицю з абсолютними частотами (АЧ) всіх слів, часто не включаючи в неї найпоширеніші слова (їх ще називають стоп-словами; у джерелі наводять англійські артиклі *a, an, the*, а для української мови це можуть бути, наприклад, сполучники *і, та, а, але*). Далі вираховують вагу для кожного речення (Sentence Weight (SW)) на основі нормалізованих довжиною речень АЧ з таблиці. Нас цікавлять речення з найбільшими значеннями SW, які і складатимуть спрощений текст. Можна використати порогове значення для збереження певної кількости таких речень (бо вони вже є спрощеним текстом; наприклад, потрібен спрощений текст з 5-ти речень, тоді програма вибирає 5 речень з найбільшим значенням SW) [26].

Значення TF-IDF вираховується за формулою

$$\text{TF-IDF (w)} = \text{TF (w)} \times \text{IDF (w)},$$

де TF (w) = АЧ слова / кіл-сть усіх слів в уривку

і IDF (w) = log (кількість усіх уривків / кіл-сть уривків із словом) [26] (w – слово).

Складно точно перекласти назву параметра TF-IDF одним або кількома словами на українську мову, тому при поясненні підходу вживається абревіатура з назви техніки Term Frequency – Inverse Document Frequency.

SW вираховується за формулою

SW (s) = сума TF-IDF (w) у межах s / кількість w в s [26] (s – речення).

Перед самими підрахунками слова в тексті піддають попередній обробці: усі великі літери в тексті перетворюють на малі, прибирають пунктуацію та стоп-слова, застосовують лематизацію до кожної словоформи [26].

Екстрактивний підхід передбачає попередню обробку для всіх слів, створення частотного розподілу їх і підрахунку SW для кожного речення,а пізніше – узяття певної кількости речень з найбільшим значенням SW і визначення їх як результату роботи програми. Таким чином утворюється спрощений текст [26].

2. Абстрактний підхід (Abstractive Approach)

В абстрактному підході для спрощення вхідного тексту відбувається генерування нового вихідного, який зберігає основний зміст, але має простішу форму. [26].

Типовими операціями в абстрактному підході є поділ речень, видалення та додавання тексту. Популярними способами застосування підходу на практиці є моделювання seq2seq (seq2seq modelling – моделі в машинному навчанні, які навчаються на паралельних корпусах [22]; спрямовані на реалізацію мовного перекладу, підпису зображень, розмових моделей, реферування тексту [27]), підходи на базі побудови

парсингових дерев, Повторювані нейронні мережі (ПНМ), Мережі короткочасної пам'яти (МКП). Попередня обробка даних включає в себе очищення тексту від розділових знаків та спеціальних символів, написання всіх слів малими буквами і складання словника словоформ. Потім вхідні речення векторизуються в числову форму для моделі з однорідним вектором довжини шляхом усічення або заповнення. Як тільки модель натренована, програма з такою моделлю спочатку векторизуватиме будь-який новий текстовий екземпляр, далі перетворюватиме його на рівномірну довжину, а потім спрощуватиме його за допомогою моделі перед тим, як конвертувати в текстову форму з числової. Коротко це можна пояснити так: вхідний текстовий екземпляр → векторизація → рівномірна довжина в числовій формі → симпліфікація → вихідний текстовий екземпляр [26].

Абстрактний підхід можна дуже узагальнено назвати лексичною симпліфікацією або генеруванням нового тексту. Підхід має надзвичайно багато методів обробки тексту, тож їхня кількість не дозволяє описати їх усіх, у рамках цього підрозділу подано лише узагальнену інформацію [26].

Програми, у логіці яких задіяні ШНМ, використовуються в різних сферах життя людини, у тому числі NLP. Їхня перевага в тому, що вони здатні самостійно навчатися на тренувальних сетах і давати кращі результати, ніж, наприклад, програми з написаними вручну правила. Застосування ШНМ в NLP бере свій початок у 2001 р. і за ці роки встигло дати різні способи розв'язків мовних завдань, що доцільно враховувати при

спробах написання програми симпліфікації текстів. За кількістю шарів і способом зв’язку нейронів існує 6 видів ШНМ. При застосуванні ШНМ для симпліфікації текстів можуть бути використані екстрактивний і/або абстрактний підходи, що тяжіють до реферування та генерування тексту відповідно. Вони пропонують різні варіанти спрощення текстів, поєднання яких є оптимальним шляхом для створення ефективної системи.

РОЗДІЛ 3

СТВОРЕННЯ ПРОГРАМИ АВТОМАТИЧНОЇ СИМПЛІФІКАЦІЇ ТЕКСТІВ ПРО ТВАРИН УКРАЇНСЬКОЮ МОВОЮ

3.1. Особливості моделі спрощення текстів

Створення програми автоматичної симпліфікації українськомовних текстів про тварин є комплексним завданням, що вимагає чимало підготовчих заходів. На сьогодні написано модель машинного навчання, яка тренується на спрощених вручну текстах з Вікіпедії. У цьому підрозділі подано основну інформацію про її будову, функціонування, використані навчальні дані тощо.

Як було зазначено в одному з попередніх розділів, програми можуть здійснювати спрощення, базуючись на створених розробниками правилах або лише тренуючись на попередньо оброблених (вручну або автоматично) навчальних даних, або поєднуючи тренування на навчальних даних та

виконання прописаних авторами [програми симпліфікації] правил. Створена модель є неконтрольованою (Unsupervised), тобто такою, яка самостійно шукає зв’язки між вхідними текстами та їхніми спрощеними варіантами і так навчається спрощувати нові тексти, що користувачі даватимуть на вхід. Такий підхід вибрано через неможливість передбачити всі ймовірні варіанти речень, які вона спрощуватиме в майбутньому, і ситуації, коли написані вручну правила діятимуть не так, як було задумано (спотворюватимуть зміст вхідного тексту, ускладнюватимуть його тощо).

Для ефективного здійснення симпліфікації мовні моделі потребують якнайбільше зібраних та придатних для тренування навчальних даних. У цій роботі було застосовано спрощені вручну статті про тварин з Вікіпедії (усього 37 057 слововживань (16 статей)), спочатку розміщені у docx-файлах, пізніше перенесені в xlsx-таблицю з колонками “ДО” та “ПІСЛЯ”. Кожне речення та кожен абзац оригінальних текстів були занесені до колонки “ДО”, а результат їхнього спрощення – до колонки “ПІСЛЯ”. Під час занесення до таблиці речень разом з їхніми обробленими варіантами підраховано, що у процесі ручного спрощення 53 речення було вилучено, 824 речення – перетворено без збільшення кількости речень (824 рази одне речення залишалось одним реченням), 270 речень – перетворено на 2 речення (270 разів з одного речення утворено два більш легких для сприйняття), 89 речень – на 3, 16 речень – на 4, 6 речень – на 5. У 7 ситуаціях було складно точно визначити, чи коректно називати певний уривок реченням (найчастіше це виникало при написанні переліків), тому ці ситуації вважатимемо за невизначені. У результаті до таблиці занесено 1264 речення. Пізніше ухвалено рішення про надання моделі даних з більшим контекстом, чого можна досягнути, додавши до навчальних даних абзаци з результатами спрощення їх. Як наслідок, таблиця розширилася до 1569

рядків і згодом була конвертована в json-файл, з яким працює створена в ході роботи модель.

Після зібрання даних та очищення їх від дублікатів та порожніх комірок (якщо речення чи абзац вилучено під час ручного спрощення, у комірках колонки “ПІСЛЯ” ставилася крапка, щоб вони не були порожніми) можна переходити до вибору типу моделі. Оскільки існує багато підходів до створення моделей машинного навчання для реалізації завдань NLP, неможливо точно стверджувати, який саме буде ефективним у кожному окремому випадку. Проте в цій роботі вибрано модель типу BERT, яку написано мовою програмування Python 3.10.5 у середовищі Visual Studio Code 1.78.2.

Модель BERT (Bidirectional Encoder Representations from Transformers) – мовна модель, представлена розробниками Google у 2018. Основною особливістю її роботи є використання попередньо підготовленої мети “маскованої мовної моделі” (“masked language model” (MLM)). Маскована мовна модель випадковим чином маскує деякі токени з вхідних даних (токенами зазвичай є слова – послідовності текстових символів від пробілу до пробілу) і має на меті передбачення початкового ідентифікатора словника замаскованого слова, спираючись лише на його лівий і правий контекст. Також ця модель виконує завдання “передбачення наступного речення”, що дозволяє їй тренуватися на парах текстових даних [12].

Реалізувати модель BERT можна за допомогою вбудованих Python-бібліотек: nltk, sklearn, json, pandas, transformers, torch, numpy, tqdm. Перед прописанням основної логіки програми потрібно імпортувати їх та/або їхні модулі. Далі сформовано список стоп-слів, які будуть вилучені при токенізації вхідних речень та абзаців. Оскільки не вдалося завантажити готовий перелік українських стоп-слів з бібліотеки nltk, довелося написати свій. Для досягнення об’єктивности до списку занесено лише сполучники

та займенники-прикметники (займенники прикметникового типу на зразок “той”, “такий” тощо). Після цього проголошено використання графічного процесора GPU для прискорення навчання моделі.

На цьому етапі в коді відкривається json-файл у режимі читання, звідки завантажуються дані. Створюються 2 списки, елементами яких є комірки з колонок “ДО” та “ПІСЛЯ”. Звернення до класу DataFrame бібліотеки pandas дозволяє створити об’єкт цього класу, який містить два стовпці з назвами “ДО” та “ПІСЛЯ”, у яких є відповідні дані з двох створених трохи раніше списків. По тому кожне слово в обох колонках перетворено на нижній регістр.

Наступним кроком є токенізація текстових даних. Спочатку здійснюється токенізація на речення, тому що це дозволятиме більш точно аналізувати синтаксичну структуру текстових даних, але оскільки аналіз кожного окремо взятого слова дозволить детальніше дослідити контекст його [кожного слова] вживання, пізніше всередині кожного речення здійснюється токенізація на слова. Між цими двома діями здійснюється злиття вкладених списків речень в один плоский список (створюється список, у якому кожен елемент є окремим реченням зі списку речень, створеного на моменті токенізації на речення). Описані дії відбуваються з даними з обох колонок по черзі. Після токенізації два списки з токенізованими даними об’єднуються в один список, у якому елементами є пари з обох колонок у вигляді кортежів (пара є кортежем).

На цьому етапі можна переходити до створення моделі. Спершу створюється екземпляр класу BertTokenizer з використанням моделі bert-base-uncased. Потім здійснюється кодування тексту за допомогою токенізатора BERT: за допомогою опцій add_special_tokens, return_attention_mask, pad_to_max_length, max_length, return_tensors метод

batch_encode_plus створює словник encoded_data, наповнений потрібною для реалізації моделі інформацією.

Далі відбувається підготовка міток класів та вхідних даних для моделі BERT: labels (містить мітки класів, що визначені в колонці "ПІСЛЯ" (об'єкт df); метод fit_transform з класу LabelEncoder перетворює текстові мітки класів у числовий формат), input_ids (містить закодовані токени зі словника encoded_data, тобто послідовність числових ідентифікаторів, які представляють різні токени у тексті), attention_masks (містить дані, які вказують моделі BERT, які токени слід враховувати при обчисленні уваги; є послідовністю чисел, де 1 вказує на наявність токена, а 0 - на його відсутність (якщо токен відсутній, на його місце ставиться інший токен, що називається *заповнювач*), min_length (містить мінімальну довжину серед трьох даних: довжини input_ids, довжини labels і довжини attention_masks, що використовується для забезпечення однакової довжини цих даних перед подальшою обробкою), input_ids, labels, attention_masks (обрізаються до мінімальної довжини min_length, щоб забезпечити однакову довжину усіх трьох даних, що робиться для існування відповідности між вхідними даними та мітками класів при подальшому навчанні та тестуванні моделі).

Після цього здійснюється поділ даних на тренувальну, тестову та валідаційну вибірки для моделі BERT. Метод train_test_split розділяє вхідні дані (input_ids), мітки класів (labels) і дані уваги (attention_masks) на тренувальну та комбіновану вибірки для валідації та тестування. random_state=42 означає початкове значення генератора випадкових чисел для повторюваности результатів, test_size=0.37 вказує, що розмір тестової вибірки становить 37% від загального набору даних, а shuffle=True показує, що дані перемішуються перед розділенням. Значення input_ids, labels і attention_masks розділяються на дві групи - val_test (валідація + тестування) і train (тренування). Розмір val_test групи становить 37% від загального

набору даних, а розмір train групи складається з решти 63% даних. Розділення відбувається таким чином, що дані для валідації та тестування розміщуються у змінних в val_test_inputs, val_test_labels і val_test_masks, а дані для тренування - у змінних train_inputs, train_labels і train_masks. Потім метод train_test_split застосовується повторно, але до val_test_inputs, val_test_labels, val_test_masks та з іншими опціями: test_size=0.5 означає, що розмір кожної з груп на val (валідація) і test (тестування) становить по 50% від val_test групи; значення random_state та shuffle такі самі, як і при першому застосуванні методу. Результати роботи методу train_test_split під час другого застосування зберігаються у змінних val_inputs, test_inputs, val_labels, test_labels, val_masks і test_masks.

Далі потрібно зробити перетворення даних з формату Python на формат torch.tensor, що потрібно для функціонування моделі BERT. Кожна змінна (train_inputs, val_inputs, test_inputs, train_labels, val_labels, test_labels, train_masks, val_masks, test_masks) перетворюється зі звичайного Python-об'єкту у відповідний тензор PyTorch за допомогою методу tensor. Це дозволяє здійснювати операції над даними з використанням функцій та методів, які надає PyTorch.

Потім слід створити датасети та даталоадери для тренування, валідації та тестування моделі. Кожен датасет (train_data, val_data, test_data) створюється з вхідних даних (train_inputs, val_inputs, test_inputs), даних уваги (train_masks, val_masks, test_masks) та міток класів (train_labels, val_labels, test_labels) за допомогою класу TensorDataset з бібліотеки PyTorch. Кожен даталоадер (train_dataloader, val_dataloader, test_dataloader) створюється зі створених раніше датасетів (train_data, val_data, test_data) за допомогою класу DataLoader з бібліотеки PyTorch. batch_size = 16 означає, що під час тренування, валідації та тестування моделі будуть використовуватись пакети даних розміром 16 зразків кожен.

Після описаних кроків потрібно зробити завантаження та налаштування моделі BERT для класифікації тексту. Можна здійснити це за допомогою методу from_pretrained, де 'bert-base-uncased' означає попередньо навчену версію моделі BERT, яка використовує нижній регістр для токенів. num_labels=1 означає кількість класів, яку модель має враховувати при здійсненні класифікації тексту. Рядок model.to(device) переносить модель на обчислювальний пристрій, встановлений раніше в рядку device = torch.device('cuda'). Це означає, що обчислення будуть виконуватись на GPU (якщо доступно) для прискорення обробки даних.

Потім переходимо до налаштування оптимізатора та критерію для тренування моделі. Оптимізатор torch.optim.AdamW обирається для оптимізації параметрів моделі. model.parameters() означає, що оптимізатор буде оновлювати ваги моделі під час тренування, lr=1e-5 означає швидкість навчання (learning rate) для оптимізатора. Критерій torch.nn.CrossEntropyLoss() обирається для визначення функції втрати (loss function) між прогнозованими значеннями моделі та істинними мітками класів. Визначення відбуватиметься під час тренування моделі.

Ми готові перейти до тренування та тестування моделі. Зовнішній цикл for epoch in range(epochs): виконується протягом заданої кількості епох (epochs). Усередині кожної епохи модель встановлюється у режим тренування за допомогою методу train(). Внутрішній цикл for batch in progress_bar: проходить через тренувальний даталоадер train_dataloader і навчає модель на пакетах даних. У циклі виконуються наступні кроки: 1) Дані пакета переносяться на пристрій GPU (якщо доступний) за допомогою t.to(device) для кожного елемента пакета. 2) Градієнти оптимізатора обнуляються з викликом optimizer.zero_grad(). 3) Модель отримує вхідні дані та виконує передачу вперед з викликом outputs = model(inputs, attention_mask=masks, labels=labels.float()). 4) Обчислюється втрата (loss) і

прогнози моделі (logits). 5) У train_loss накопичується врата. 6) Виконується зворотній прохід (loss.backward()) та оптимізатор оновлює ваги моделі (optimizer.step()). 7) У прогрес-барі виводиться інформація про втрату тренування.

Після завершення тренування кожної епохи модель переходить у режим оцінювання (model.eval()). Внутрішній цикл for batch in progress_bar: проходить через тестовий даталоадер test_dataloader для оцінки моделі на тестових даних. У циклі виконуються наступні кроки: 1) Дані пакета переносяться на пристрій GPU (якщо доступний) за допомогою t.to(device) для кожного елемента пакета. 2) З вимкненим відстеженням градієнтів (torch.no_grad()), модель отримує вхідні дані та виконує передачу вперед. 3) Обчислюється втрата (loss) і прогнози моделі (logits). 4) Втрата накопичується у test_loss. 5) Обчислюється кількість правильних прогнозів (correct_predictions) та загальна кількість прогнозів (total_predictions). 6) Виводиться інформація про втрату тестування та точність. Після закінчення кожної епохи друкується інформація про втрату тренування, втрату тестування та точність моделі [Додаток 11].

При запуску програми було отримано такі результати: 1) Епоха 1 – Training Loss: 2092345,6531, Test Loss: 743164,2928, Accuracy: 0,34%, 2) Епоха 2 – Training Loss: 2077314,5594, Test Loss: 738745,5016, Accuracy: 0,34%, 3) Епоха 3 – Training Loss: 2069550,0172, Test Loss: 735722,7089, Accuracy: 0,34%, 4) Епоха 4 – Training Loss: 2062339,8953, Test Loss: 733555,8174, Accuracy: 0,34%, 5) Епоха 5 – Training Loss: 2053890,9672, Test Loss: 731900,2582, Accuracy: 0,34%, 6) Епоха 6 – Training Loss: 2049424,4563, Test Loss: 730552,1776, Accuracy: 0,34% [Додатки 12, 13].

Оскільки робота програми виявилася тривалою, вирішено взяти 6 епох. З результатів видно, що при кожній наступній ітерації втрати тренування та тестування зменшуються, але точність лишається такою

самою. Це може відбуватися або через недостатню складність моделі (модель досягнула певного плато в точності, де вона не покращується, навіть коли втрати зменшуються), або через недостатню кількість тренувальних даних, або через неправильну оцінку точности, або через некоректне обчислення точности.

Отже, для спрощення українськомовних текстів про тварин створено модель неконтрольованого навчання типу BERT, яка тренується на текстових даних, розміщених у json-форматі, обсягом 1569 одиниць. Навчальні дані були створені вручну шляхом спрощення текстів з Вікіпедії. Код програми реалізований мовою Python (з використанням вбудованих бібліотек для обробки природної мови, роботи з табличними даними, побудови моделі BERT) у середовищі Visual Studio Code. Через великий обсяг пам’яти, що застосовується під час запуску програми, тренування та тестування моделі відбуваєься 3 рази. Наразі для подальшого прогресу у створенні програми автоматичного спрощення українськомовних текстів про тварин слід зосередитися на пошуку та усунення причини низької точности роботи моделі.

3.2. Ідеї покращення роботи програми

Описана модель потребує змін для підвищення ефективности своєї роботи. Тому в цьому підрозділі буде розглянуто ідеї покращення наявного коду для наближення до створення програми автоматичної симпліфікації текстів про тварин українською мовою.

На початку реалізації програми було вибрано підхід машинного навчання, тобто навчання на текстових даних з метою пошуку моделлю зв’язків між оригінальними та спрощеними текстами і подальшого

використання їх під час симпліфікації текстів, наданих користувачем. Але застосування цього підходу не виключає можливість написання власних правил у вигляді окремих функцій, які би застосовувалися до вхідних текстів. Проте слід враховувати, що, по-перше, правила не мають погіршити якість роботи моделі, тому при введенні кожного правила потрібно перевіряти, як воно взаємодіє з цілою програмою, і, по-друге, запровадження правил має поєднуватися з регулярним збільшенням обсягу навчальних даних.

Оскільки тексти для ручного спрощення беруться з Вікіпедії, де кожен користувач може опублікувати власну статтю без попереднього редагування, у них часто є багато помилок. З лексичними можна спробувати впоратися таким чином: скласти словник найпоширеніших кальок і навчити програму перевіряти кожне слово, порівнюючи його з вмістом словника, і за потреби виправляти його на літературний відповідник. Якщо помилка буде пов’язана з уживанням слова в невластивому йому значенні, доведеться застосувати контекстний аналіз, що є складнішою процедурою. Стосовно граматичних помилок, можна так само скласти словник з правильними узгодженнями дієслів (в узгодженні дієслів найчастіше трапляються помилки на зразок *наслідувати актрисі*, коли правильно *наслідувати актрису*) та інших частин мови, які викликають труднощі і теж навчити програму знаходити їх у текстах і виправляти на правильні варіанти.

Додатково потрібно запровадити норми правопису 2019-ого року. Якщо із нормами на зразок слова *проєкт* (і похідних від нього) можна впоратися, занісши це слово до словника за таким самим принципом, як лексичні та граматичні помилки, то із закінченням –и в родовому відмінку для іменників 3 відміни (радости, любови) можна впоратися за допомогою вбудованої Python-бібліотеки pymorphy2, написавши рядки, де сказано, що

слова з тегами nomn (називний відмінок), femn (жіночий рід), sing (однина) та нульовою флексією у формі з тегами gent (родовий відмінок), femn (жіночий рід), sing (однина) повинні мати флексію –и. Аналогічно потрібно вчинити з усіма іншими чинними нормами.

Для того, щоб досягнути гендерної рівности в текстах, біля іменників (окрім назв тварин), прикметників, порядкових числівників, займенників-прикметників можна додати в дужках закінчення жіночого роду. Втілити цю ідею допоможе pymorphy2: біля слів з тегами masc (чоловічий рід) та sing (однина) дописати флексію жіночого роду у форматі (–*буква*) відповідно до тегу відмінка. В ідеалі варто не тільки писати закінчення в дужках, а квазіфлексію, утворену злиттям суфікса та закінчення («Асоціація ***виробників (-ць)*** товарів для домашніх тварин (…)» (Пес свійський, Текст 22, 1-ий абзац [Додаток 14]), «При оцінці собак ***фахівці (-чині)*** враховують конституцію собак» (Пес свійський, Текст 16, 1-ий абзац [Додаток 15])). До займенників-іменників простіше додати назву жіночого роду в дужках, склавши перед тим словник з парами таких займенників («***Він (вона)*** [дресирувальник (-ця)] має знати рухи (…)» (Пес свійський, Текст 12, 2-ий абзац [Додаток 15])).

Норми милозвучности хоч і не є першочерговим завданням майбутньої програми, але їх також потрібно дотримуватися. У деяких ситуаціях складно визначити, який прийменник – *у* чи *в* – буде доречнішим, тому достатньо спиратися на норми українського правопису. Те саме стосується і правил чергування сполучників *і-та-й*.

Можна спробувати покращити ефективність роботи моделі, написавши правило синтаксичної симпліфікації. Наприклад, знайти спосіб визначення виду(-ів) зв’язку складного речення і, залежно від виду(-ів), здійснювати спрощення: якщо зв’язок сурядний єднальний, можна ділити предикативні частини на окремі речення; якщо сурядний протиставний –

так само, але зі збереженням сполучника на початку другого речення; якщо сурядний розділовий – можна поділити, при цьому кожну предикативну частину, окрім першої, починати зі сполучників *або*; якщо підрядний означальної семантики – визначити, до якого слова в головній частині відноситься сполучний засіб, замінити його на це слово і переставити після дієслова, якого воно стосується і якщо воно є додатком; якщо підрядне супровідне речення – так само; якщо підрядний з'ясувальної семантики – можна лишити без змін; якщо підрядний обставинної семантики – так само; якщо недиференційований – перетворити на речення з підрядним обставинним.

Коли програма зможе спрощувати тексти, взаємодіючи з користувачем через консоль, потрібно створити більш комфортний для користувачів простір взаємодії з програмою. Можна зробити це шляхом написання сайту мовою розмітки HTML з інтуїтивно зрозумілим інтерфейсом: поле для введення тексту, що потребує спрощення (з інструкцією поряд щодо обмежень на кількість символів у вхідному тексті), поле для результатів, куди користувачі не зможуть дописувати, кнопка з написом "Спростити", прогрес-бар, який показує, що симпліфікація триває. Якщо процес перетворення виявиться довготривалим, варто спробувати навчити програму передбачати хоча б приблизний час, потрібний на виконання симпліфікації, і писати його поряд з прогрес-баром. Слід наголосити на важливості прогрес-бару, оскільки коли користувачі не бачать, що Інтернет-сторінка отримала їхній запит і займається його обробкою, вони починають або багаторазово тиснути на кнопку з бажаною дією, або перезавантажувати сторінку, що лише сповільнює виконання запиту.

Таким чином, існують шляхи для підвищення ефективности роботи наявної моделі. Поєднання машинного навчання з прописаними вручну

правилами є оптимальним підходом при розв'язку завдань NLP, зокрема спрощення, але при введенні у програму готових правил потрібно перевіряти, як вони взаємодіють з нею і як при цьому змінюється ефективність роботи моделі. Також важливим є створення платформи для взаємодії користувачів з програмою. Найпростійшим варіантом є Інтернет-сторінка, написана мовою розмітки HTML, з компактним та інтуїтивно зрозумілим інтерфейсом. Описані в цьому підрозділі кроки мають наблизити існування ефективної системи автоматичної симпліфікації текстів про тварин українською мовою.

ВИСНОВКИ

Дослідивши різноманітні методи синтаксичної та лексичної симпліфікації текстів у різних мовах, застосування штучних нейронних мереж у NLP, створивши модель машинного навчання для спрощення українськомовних текстів про тварин, можна сформулювати такі висновки.

Симпліфікацією називають процес перетворення тексту, у результаті якого він [текст] стає легшим для сприйняття людьми з низькими навичками читання, але при цьому зберігає найважливішу інформацію. Використовують 3 основні підходи для реалізації цього завдання: створені вручну правила, навчання на попередньо підготовлених даних, поєднання описаних способів. Симпліфікація текстів може здійснюватися не лише для людей, але і для розв'язання певних мовних завдань. Розділяють методи синтаксичної та лексичної симпліфікації, при цьому для реального спрощення спочатку потрібно виконати синтаксичну і тільки потім – лексичну. Створено системи автоматичної симпліфікації для найпоширеніших мов: китайської (CLS), англійської (Biran et al. [2011],

Glavaš and Štajner [2015]), іспанської (LexSiS) тощо. Найбільш популярною в цій темі мовою можна назвати англійську.

Алгоритми глибокого навчання дозволяють покращувати результати завдань з обробки природної мови, зокрема спрощення текстів. Робота цих алгоритмів передбачає створення та застосування штучних нейронних мереж (ШНМ), що імітують роботу людського мозку та пов’язані з поняттями сильного і слабкого штучного інтелекту, тесту Тюрінга. Поява ШНМ в NLP бере свій початок у 2001 р. За ці роки було створено кілька видів ШНМ, які, маючи різну кількість шарів і способів з’єднання вузлів, можуть виконувати різні завдання NLP, у тому числі здійснювати симпліфікацію текстів. Оскільки ця операція поєднує в собі ознаки текстового реферування та генерування текстів, при застосуванні ШНМ для симпліфікації можуть бути використані екстрактивний (близький до реферування) і/або абстрактний (близький до генерування) підходи.

Завдання, зазначені в роботі, повністю виконано:

1) подано визначення симпліфікації та описано цілі й основні підходи здійснення її;
2) досліджено та описано сучасні методи синтаксичної симпліфікації;
3) досліджено та описано актуальні методи лексичної симпліфікації;
4) окреслено основні етапи історії застосування штучних нейронних мереж у NLP;
5) описано класифікацію штучних нейронних мереж;
6) досліджено підходи здійснення реферування та генерування текстів, що можуть бути корисні при реалізації симпліфікації;
7) описано особливості створеної моделі спрощення текстів;
8) надано ідеї для підвищення ефективности роботи моделі.

У ході виконання практичної частини виявилося, що створення, навчання і тестування моделі, яка є основою програми автоматичної

симпліфікації українськомовних текстів про тварин, є складним і затребуваним у часи інформаційного перенавантаження завданням. Одним з найбільш великих за обсягом роботи кроків можна назвати збір навчальних даних і приведення їх у придатний для тренування моделі вигляд. Якщо вбудовані в Python бібліотеки (для обробки природної мови, роботи з табличними даними, побудови моделі BERT) пропонують готові зразки створення мовних моделей, неможливо прискорити накопичення даних для навчання, адже ці дані підготовлені вручну. Запропоновано ідеї підвищення ефективности роботи моделі та її взаємодії з користувачами, відповідно, дослідження є перспективним і може бути продовжено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. O. Abend, A. Rappoport, E. Sulem. Simple and Effective Text Simplification Using Semantic and Neural Methods. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*. 2018. Vol. 1. P. 162-173.
2. Barlacchi G. ERNESTA: A sentence simplification tool for children's stories in Italian / G. Barlacchi, S. Tonelli. – volume 2 of CICLING, 2013. – 476–487 p. – In Proc. of the 14th International Conference on Intelligent Text Processing and Computational Linguistics.
3. Empirical identification of text simplification strategies for reading-impaired people / [Bautista S., Gervás P., Hervás R., León C.]. – the Netherlands, Maastricht, In European Conference for the Advancement of Assistive Technology, Maastricht ,2011. – 8 p.
4. Bautista S. Making numerical information more accessible: The implementation of a numerical expression simplification system for

Spanish / S. Bautista, H. Saggion. – ITL International Journal of Applied Linguistics, 2014. – 165 (2) 229–323 p.

5. Introduction to WordNet: An on-line lexical database. International Journal of Lexicography / [Beckwith R., Fellbaum C., Gross D. etc] 1990. – 87 p.
6. Y. Bengio, K. H. Cho, J. Chung, C. Gulcehre. Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *NIPS 2014 Deep Learning and Representation Learning Workshop*. 2014. 9 p.
7. Biran O. Putting it simply: A context-aware approach to lexical simplification [article] / [O. Biran, S. Brody, N. Elhadad] – [In Proc. of the 49th Annual Meeting of the Association for Computational Linguistics] – ACL, 2011 – 496-501 p.
8. Can Spanish be simpler? LexSiS: Lexical simplification for Spanish / [Bott S., Drndarević B., Rello L., Saggion H.]. – Mumbai. : COLING, 2012. – 357–374 p. – Proc. of the 24th International Conference on Computational Linguistics.
9. Design and Annotation of the First Italian Corpus for Text Simplification / [Brunato D., Dell'Orletta F., Montemagni S., Venturi G.]. – USA, Colorado, Denver : ACL, 2015. – 10 p. – (in Proceedings of The 9th Linguistic Annotation Workshop).
10. Practical Simplification of English Newspaper Text to Assist Aphasic Readers / [Canning Y., Carroll J., Delvin S. etc] – In Proc. of AAAI-98 Workshop on Integrating Artificial Intelligence and Assistive Technology, 1998. – 5 p.
11. Chandrasekar R. Motivations and methods for text simplification : [article] / [R. Chandrasekar, C. Doran, B. Srinivas] ; – In 16th International Conference on Computational Linguistics, 1996. – 1041–1044 p.

12. M.-W. Chang, J. Devlin, K. Lee, K. Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. 2019. Vol. 1. P. 4171–4186.
13. Kenneth W. Church One sense per discourse : [article] / [K. W. Church, W. A. Gale, D. Yarowsky] ; – In Proc. of the Workshop on Speech and Natural Language, 1992. – 233–237 p.
14. Cooper M. CombiNMT: An Exploration into Neural Text Simplification Models / M. Cooper, M. Shardlow. – LREC, 2020. – 7 p. – (in Proceedings of the 12th Conference on Language Resources and Evaluation).
15. De Belder J. Text Simplification for Children / J. De Belder, M.-F. Moens. – NY: ACM, 2010. – 8 p. – (in Proceedings of the SIGIR workshop on accessible search systems).
16. Dmitrieva A. Creating an Aligned Russian Text Simplification Dataset from Language Learner Data / A. Dmitrieva, J. Tiedemann. – BSNLP | EACL, 2021 – 7 p. – (in Proceedings of the 8th Workshop on Balto-Slavic Natural Language Processing).
17. Glavaš G. Simplifying lexical simplification: Do we need simplified corpora? / G. Glavaš, S. Štajner – ACL: July 26-31, 2015. – 63-68 pages (In Proc. of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing)
18. L. Hardesty. Explained: Neural networks. MIT News Office. 2017. URL: https://news.mit.edu/2017/explained-neural-networks-deep-learning-0414 .

19. Horn C. Learning a Lexical Simplifier Using Wikipedia / C. Horn, D. Kauchak, C. Manduca. – ACL, 2014. – 6 p.
20. Introducing ChatGPT. URL: https://openai.com/blog/chatgpt .
21. Jugal K. Kalita, Julian R. Medina, Daniel W. Otter. A Survey of the Usages of Deep Learning for Natural Language Processing. IEEE Transactions on Neural Networks and Learning Systems. 2019. Vol. XX, No. X. P. 1-22.
22. O. Kariuk. Unsupervised text simplification using neural style transfer. Lviv : UCU, 2020. 48 p.
23. David Kauchak. Improving text simplification language modeling using unsimplified text data / David Kauchak. – ACL, 2013 – 1537-1546 pages – (In Proc. of the 51st Annual Meeting of the Association for Computational Linguistics).
24. Lal P. Extract-based summarization with simplification / P. Lal, S. Rüger, 2002 – 8 pages – In Proc. of the Document Understanding Conference.
25. Y. Li, X. Lu, J. Qiang, Y. Yuan, X. Wu. Chinese Lexical Simplification. *IEEE/ACM TRANSACTIONS ON AUDIO, SPEECH, AND LANGUAGE PROCESSING*. 2021. Vol. XXIX. P. 1819-1828.
26. V. Mago, P. Sikka. A Survey on Text Simplification. J. ACM. 2020. Vol. XXXVII, No. IV. P. 1-26.
27. mani.wadhwa. seq2seq Model in Machine Learning. *Geeksforgeeks*. 2023. URL: https://www.geeksforgeeks.org/seq2seq-model-in-machine-learning/ .
28. Omelianchuk K. Text Simplification by Tagging / K. Omelianchuk, V. Raheja, O. Skurzhanskyi – BEA | EACL, 2021. – 15 p.
29. J.-B. Park, W. J. Park. History and application of artificial neural networks in dentistry. *European Journal of Dentistry*. 2018. Vol. XII, No. IV. P. 594-601.

30. S. Ruder. A Review of the Neural History of Natural Language Processing. *Aylien*. 2018. URL: https://aylien.com/blog/a-review-of-the-recent-history-of-natural-language-processing .

31. Saggion H. Automatic Text Simplification : [manual] / H. Saggion. – Universitat Pompeu Fabra : by Morgan & Claypool, 2017. – 137 p.

32. J. Schmidhuber. Deep Learning in Neural Networks: An Overview. Switzerland: The Swiss AI Lab IDSIA, Istituto Dalle Molle di Studi sull'Intelligenza Artificiale, University of Lugano & SUPSI, 2014. 88 p.

33. Matthew Shardlow. A Comparison of Techniques to Automatically Identify Complex Words / Matthew Shardlow. – ACL, 2013. – 7 p. – (in Proceedings of the Student Research Workshop at the 51st Annual Meeting of the Association for Computational Linguistics).

34. Shardlow M. Out in the open: Finding and categorising errors in the lexical simplification pipeline / M. Shardlow – Reykjavik, 2014. – 8 p. – (Proc. of the 9th International Conference on Language Resources and Evaluation).

35. Siddharthan A. An Architecture for a Text Simplification System / A. Siddharthan – LEC, 2002. – 8 p. – (in Proceedings of the Language Engineering Conference)

36. Siddharthan A. Syntactic simplification and text cohesion : [manual] / A. Siddharthan. – Cambridge, 2004. – 195 p.

37. Advaith Siddharthan. Syntactic Simplification and Text Cohesion / Advaith Siddharthan. – Columbia University : Department of Computer Science 2006. – 31 p.

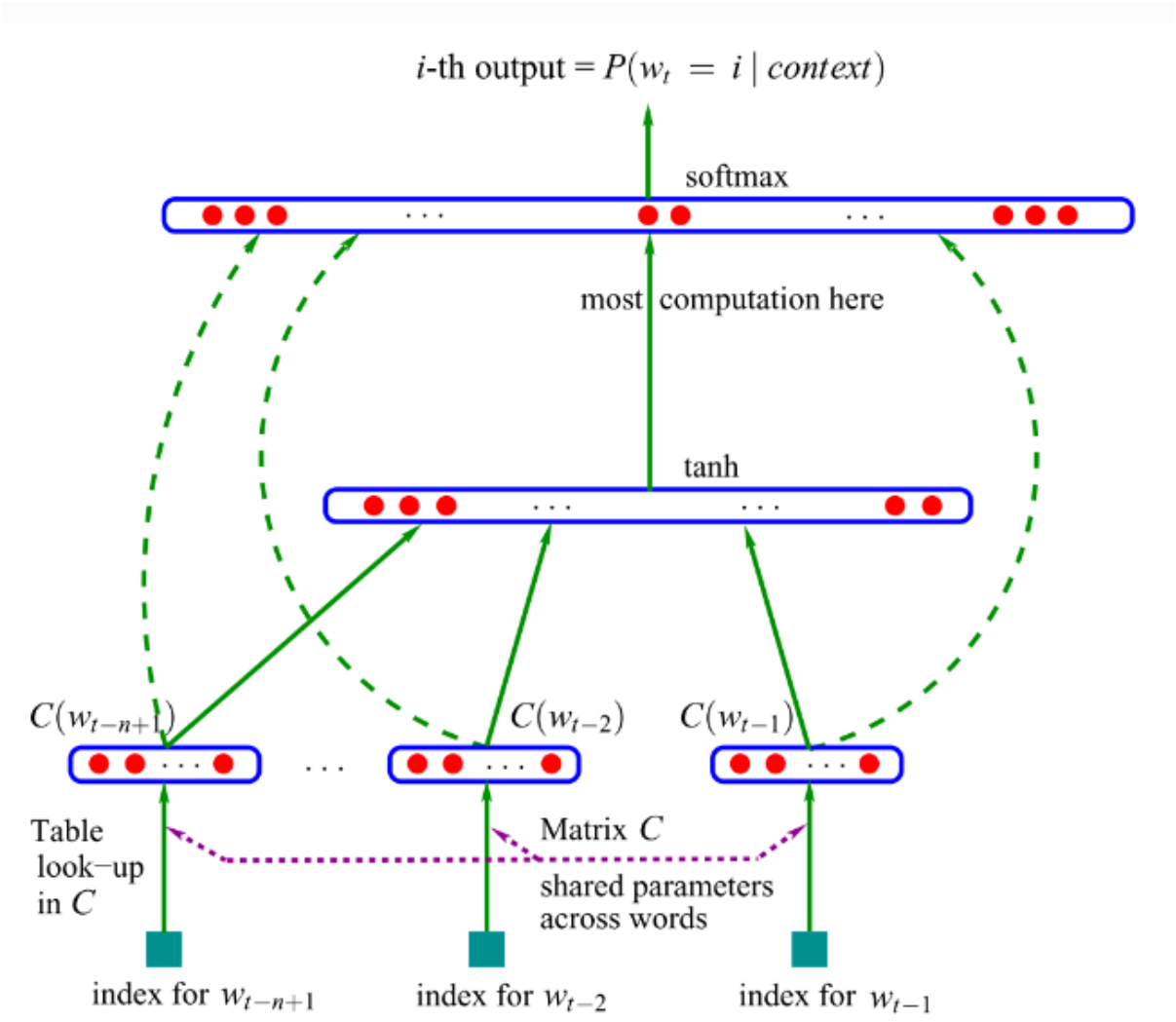
38. Advaith Siddharthan. Text Simplification using Typed Dependencies: A Comparison of the Robustness of Different Generation Strategies / Advaith Siddharthan. – ACL, 2011. – 9 p. – (in Proceedings of the 13th European Workshop on Natural Language Generation).

39. Srinivas B. Performance evaluation of supertagging for partial parsing / B. Srinivas – Boston, 1997. – 12 p. – (In Proc. of the 5th International Workshop on Parsing Technology).
40. A. M. Turing. On computable numbers, with an application to the entscheidungsproblem. The Graduate College, Princeton University, New Jersey, U.S.A., 1936. 36 p.
41. D. Yarish. Sentence Simplification in context of Automatic Question Generation. Lviv : UCU, 2019. 36 p.

ДОДАТКИ

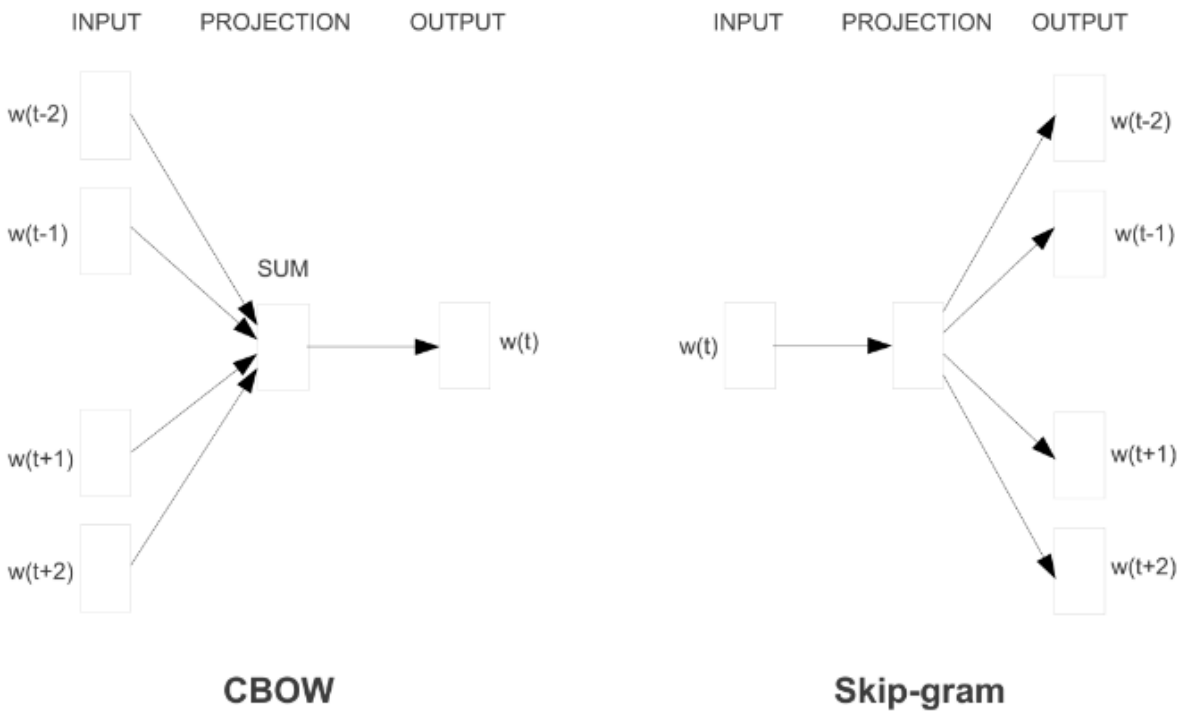
Додаток 1

Нейронна мережа Bengio et al., 2001



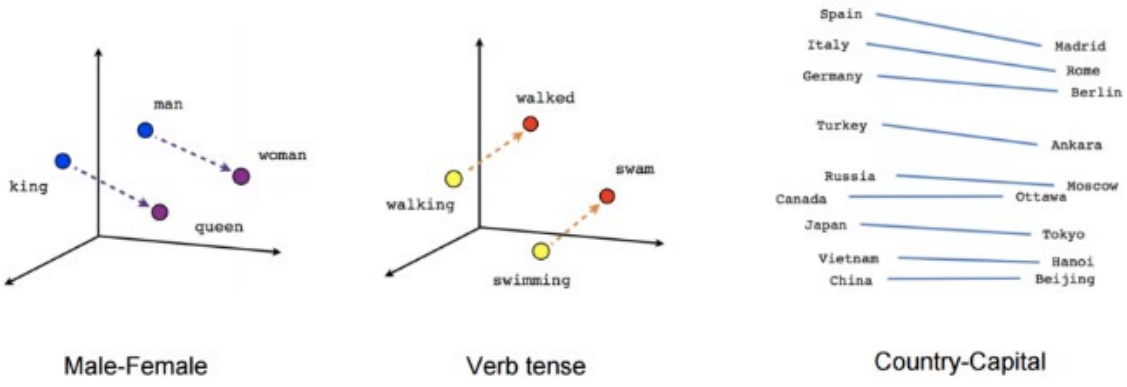
Додаток 2

Нейронна мережа Mikolov et al., 2013



Додаток 3

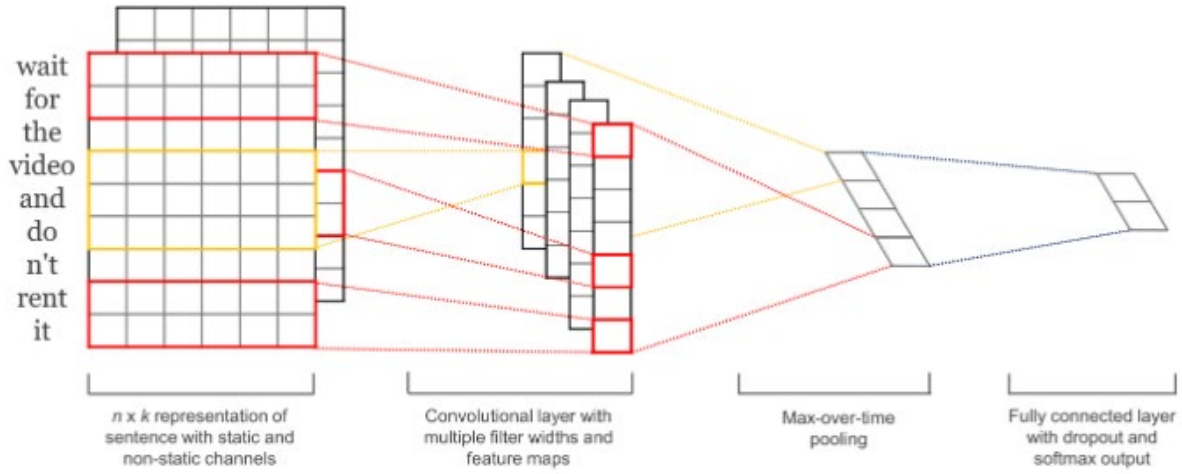
Зв’язки, утворені за допомогою алгоритму word2vec (нейронна мережа Mikolov et al., 2013)



Male-Female Verb tense Country-Capital

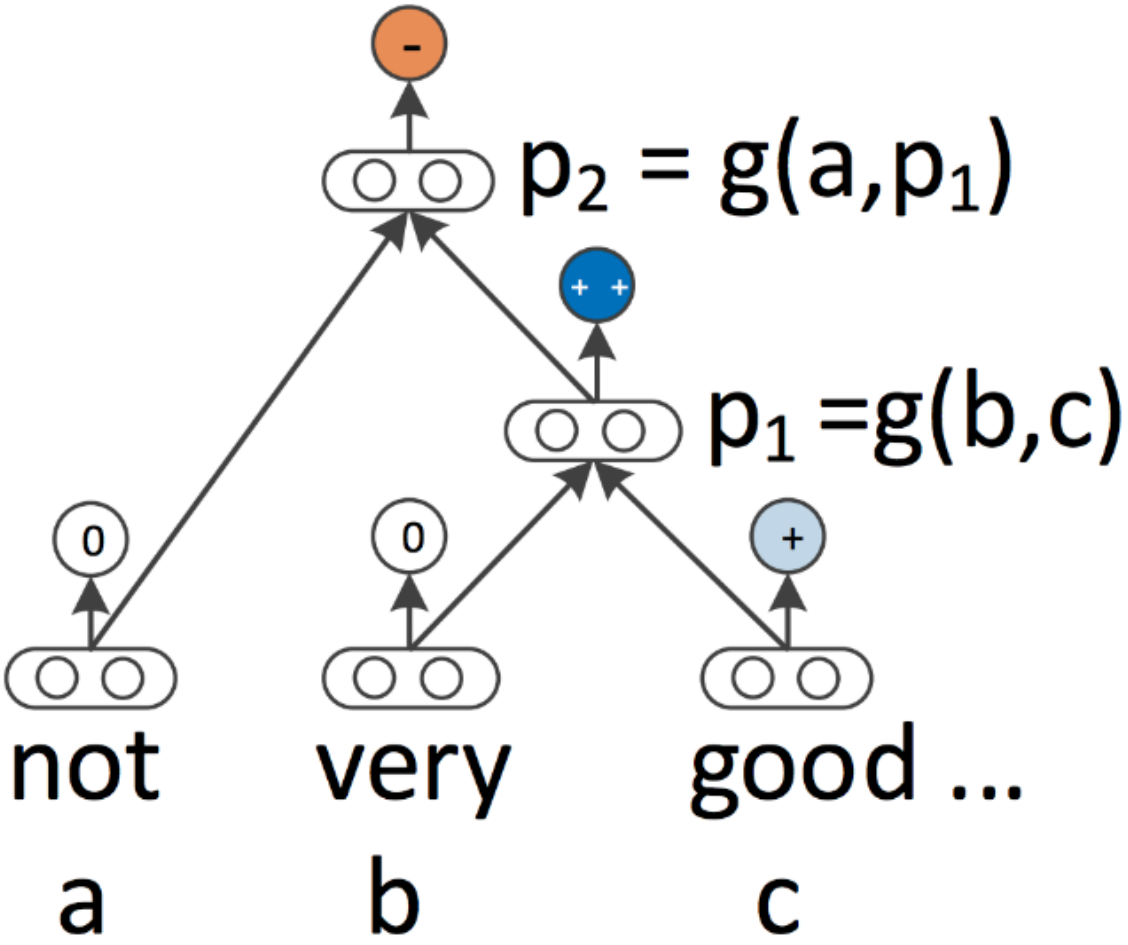
Додаток 4

Нейронна мережа Kim, 2014



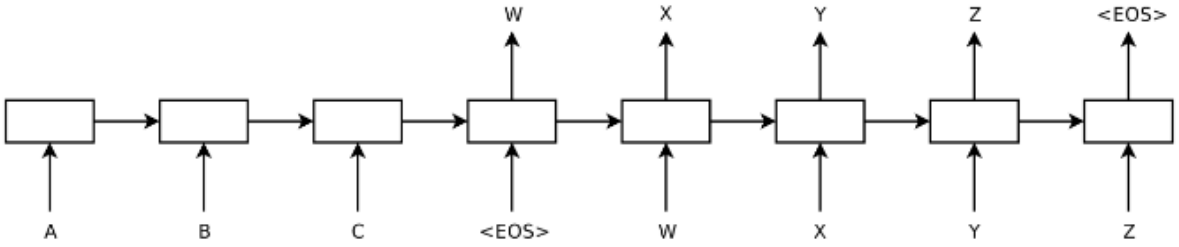
Додаток 5

Нейронна мережа Socher et al., 2013



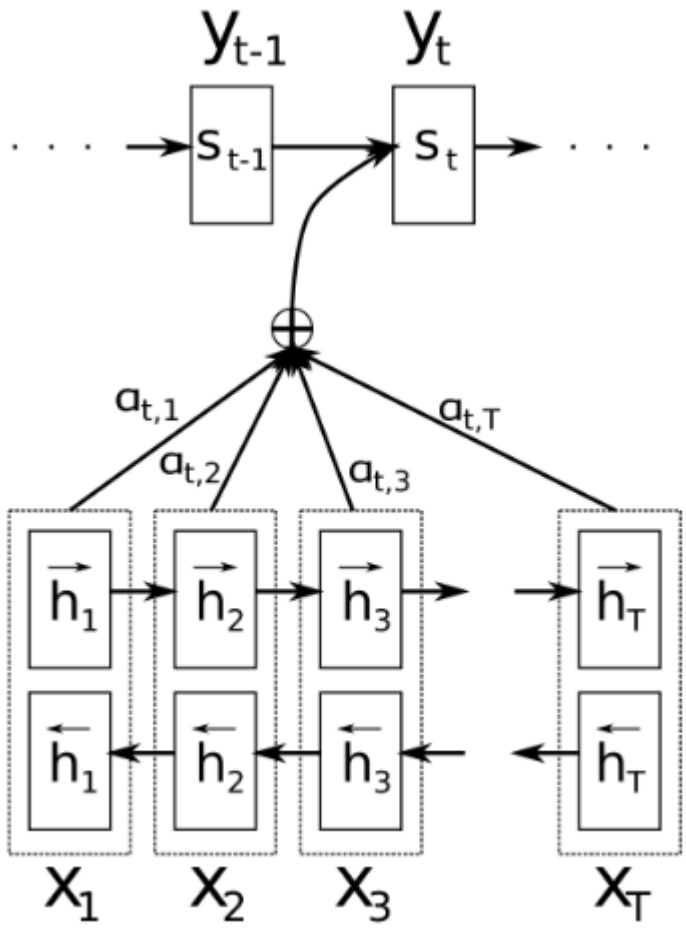
Додаток 6

ШНМ, що перетворює послідовність на послідовність (Sutskever et al., 2014)



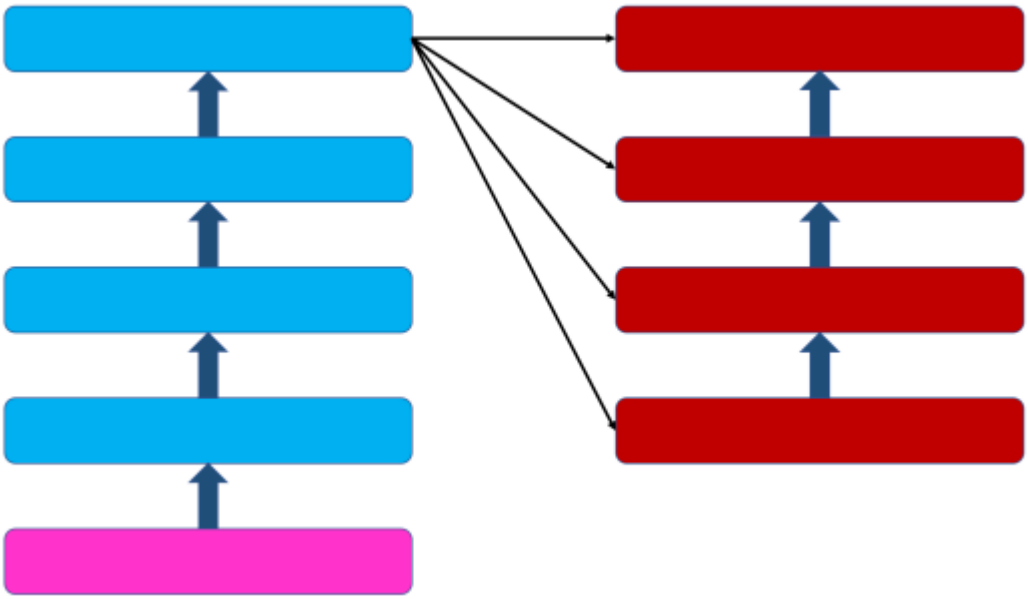
Додаток 7

ШНМ за принципом, який називається “увага” (Bahdanau et al., 2015)



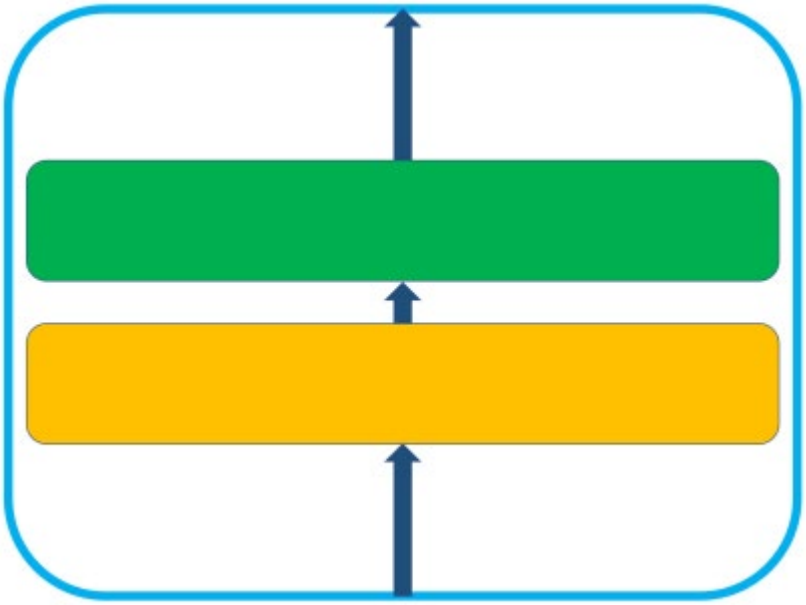
Додаток 8

Модель-трансформер: “позиційний енкодер” (рожевий), 4 енкодери (блакитні) та 4 декодери (червоні)



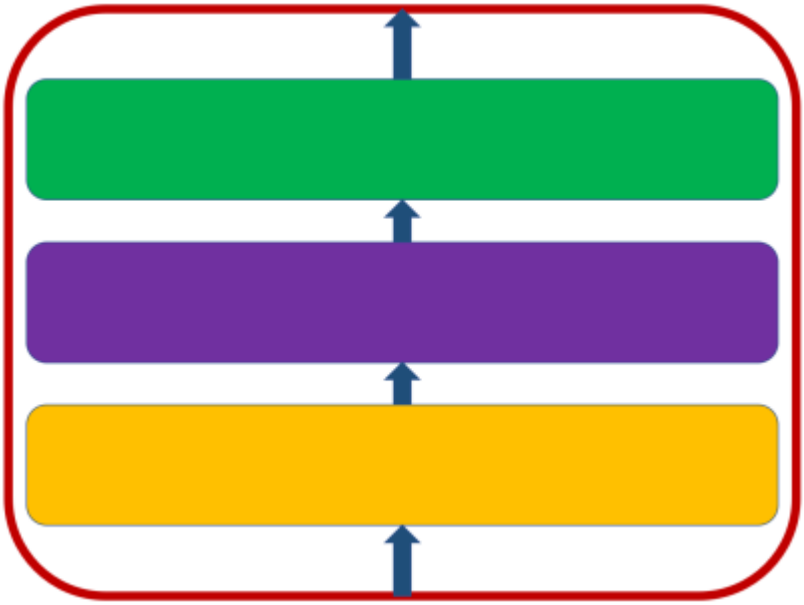
Додаток 9

ШНМ увага: внутрішня робота кожного енкодера, що містить рівень самоконтролю (жовтий), після якого йде шар прямої передачі (зелений)



Додаток 10

ШНМ увага (підвид самоуважність): внутрішня робота кожного декодера, що включає в себе шар самоуважности (жовтий), після якого йде шар уваги енкодера-декодера (фіолетовий), що завершується шаром прямої передачі (зелений)



Додаток 11

Повний код наявної на сьогодні програми

```
# Імпортування потрібних бібліотек та модулів

import nltk

nltk.download("punkt")

from nltk.tokenize import sent_tokenize, word_tokenize

from sklearn.model_selection import train_test_split

import json

import pandas as pd

from transformers import BertTokenizer, BertForSequenceClassification

import torch

from torch.utils.data import TensorDataset, DataLoader

from sklearn.preprocessing import LabelEncoder

# import numpy as np

# from sklearn.metrics import accuracy_score

from tqdm import tqdm



# Вбудований список стоп-слів

stop_words = [

    'а', 'і', 'в', 'на', 'з', 'до', 'як', 'про', 'також', 'та',

    'їх', 'його', 'але', 'або', 'у', 'від', 'по', 'не', 'за',

    'бути', 'це', 'все', 'що', 'якщо', 'коли', 'де', 'чи', 'ні',
```

```
    'той', 'цей', 'та', 'так', 'такий', 'така', 'таке', 'такі',
    'інший', 'інша', 'інше', 'інші', 'сам', 'сама', 'саме', 'самі',
    'свій', 'своя', 'своє', 'свої'
]

# print (torch.cuda.is_available())
# Використання GPU для прискорення навчання моделі (якщо доступно)
device = torch.device('cuda')

# Відкриття JSON-файлу та зчитування даних
with open('C:\\ANIMALS PROJECT\\jsonformatter.json', 'r', encoding='utf-8') as f:
    data = json.load(f)

# Створення списків, де елементами є комірки таблиці
before_list = [row['ДО'] for row in data]
after_list = [row['ПІСЛЯ'] for row in data]

# Створення DataFrame зі списків
df = pd.DataFrame({'ДО': before_list, 'ПІСЛЯ': after_list})

# Виконання попередньої обробки тексту (наприклад, перетворення на нижній регістр)
```

```
df['ДО'] = df['ДО'].str.lower()

df['ПІСЛЯ'] = df['ПІСЛЯ'].str.lower()


# Токенізація даних з колонки 'ДО'

sentences_before = df['ДО'].apply(sent_tokenize)


# Злиття вкладених списків у плоский список

sentences_before_flat = [sentence for sublist in sentences_before for sentence in sublist]


# Токенізація та видалення стоп-слів для кожного речення

tokenized_sentences_before = [word_tokenize(sentence) for sentence in
sentences_before_flat]

filtered_sentences_before = [[word for word in sentence if word.lower() not in stop_words]
for sentence in tokenized_sentences_before]


# Токенізація даних з колонки 'ПІСЛЯ'

sentences_after = df['ПІСЛЯ'].apply(sent_tokenize)


# Злиття вкладених списків у плоский список

sentences_after_flat = [sentence for sublist in sentences_after for sentence in sublist]
```

```
# Токенізація та видалення стоп-слів для кожного речення

tokenized_sentences_after = [word_tokenize(sentence) for sentence in
sentences_after_flat]

filtered_sentences_after = [[word for word in sentence if word.lower() not in stop_words]
for sentence in tokenized_sentences_after]


# Створення пар даних (оригінальні речення та їх спрощені версії)

combined_data = list(zip(filtered_sentences_before, filtered_sentences_after))


# Підготовка даних для моделі BERT

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')


# Кодування тексту за допомогою токенізатора BERT

encoded_data = tokenizer.batch_encode_plus(
    combined_data,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=512,
    return_tensors='pt'
)
```

```
# Перетворення міток класів в числовий формат
labels = LabelEncoder().fit_transform(df['ПІСЛЯ'])
input_ids = encoded_data['input_ids']
attention_masks = encoded_data['attention_mask']
min_length = min(len(input_ids), len(labels), len(attention_masks))
input_ids = input_ids[:min_length]
labels = labels[:min_length]
attention_masks = attention_masks[:min_length]

# Розділення даних на тренувальну, тестову та валідаційну вибірки
train_inputs, val_test_inputs, train_labels, val_test_labels, train_masks, val_test_masks =
train_test_split(
    input_ids, labels, attention_masks, random_state=42, test_size=0.37, shuffle=True
)

val_inputs, test_inputs, val_labels, test_labels, val_masks, test_masks = train_test_split(
    val_test_inputs, val_test_labels, val_test_masks, random_state=42, test_size=0.5,
shuffle=True
)
```

```
# Перетворення даних у формат PyTorch
train_inputs = torch.tensor(train_inputs)
val_inputs = torch.tensor(val_inputs)
test_inputs = torch.tensor(test_inputs)
train_labels = torch.tensor(train_labels)
val_labels = torch.tensor(val_labels)
test_labels = torch.tensor(test_labels)
train_masks = torch.tensor(train_masks)
val_masks = torch.tensor(val_masks)
test_masks = torch.tensor(test_masks)

# Створення датасетів та даталоадерів для тренування, валідації та тестування
train_data = TensorDataset(train_inputs, train_masks, train_labels)
train_dataloader = DataLoader(train_data, batch_size=16, shuffle=True)
val_data = TensorDataset(val_inputs, val_masks, val_labels)
val_dataloader = DataLoader(val_data, batch_size=16)
test_data = TensorDataset(test_inputs, test_masks, test_labels)
test_dataloader = DataLoader(test_data, batch_size=16)

# Завантаження та налаштування моделі BERT для класифікації
```

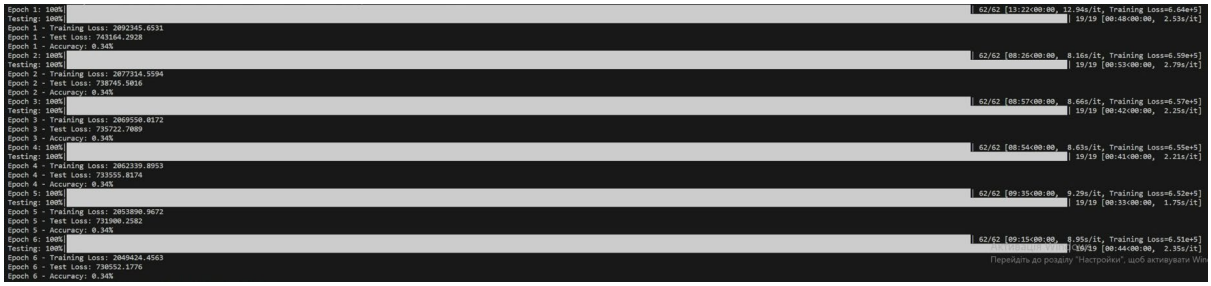
```
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=1)

model.to(device)


# Налаштування оптимізатора та критерію

optimizer = torch.optim.AdamW(model.parameters(), lr=1e-5)

criterion = torch.nn.CrossEntropyLoss()


# Тренування моделі


epochs = 6

for epoch in range(epochs):

    model.train()

    train_loss = 0.0

    progress_bar = tqdm(train_dataloader, desc=f"Epoch {epoch+1}")

    for batch in progress_bar:

        batch = tuple(t.to(device) for t in batch)

        inputs, masks, labels = batch

        optimizer.zero_grad()

        outputs = model(inputs, attention_mask=masks, labels=labels.float())

        loss = outputs.loss
```

```
    logits = outputs.logits
    train_loss += loss.item()
    loss.backward()
    torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
    optimizer.step()
    progress_bar.set_postfix({"Training Loss": train_loss / (len(progress_bar) + 1)})

# Тестування моделі
model.eval()  # Увімкнення режиму оцінювання

test_loss = 0.0
correct_predictions = 0
total_predictions = 0

progress_bar = tqdm(test_dataloader, desc="Testing")
for batch in progress_bar:
    batch = tuple(t.to(device) for t in batch)
    inputs, masks, labels = batch

    with torch.no_grad():
        outputs = model(inputs, attention_mask=masks, labels=labels.float())
```

```
            loss = outputs.loss
            logits = outputs.logits
            test_loss += loss.item()

            # Підрахунок кількости правильних передбачень
            predicted_labels = logits.argmax(dim=1)
            correct_predictions += (predicted_labels == labels).sum().item()
            total_predictions += labels.size(0)

        average_test_loss = test_loss / len(test_dataloader)
        accuracy = correct_predictions / total_predictions

    # Виведення результатів у вигляді прогрес-барів
        print(f"Epoch {epoch+1} - Training Loss: {train_loss / (len(progress_bar) + 1):.4f}")
        print(f"Epoch {epoch+1} - Test Loss: {average_test_loss:.4f}")
        print(f"Epoch {epoch+1} - Accuracy: {accuracy*100:.2f}%")
```

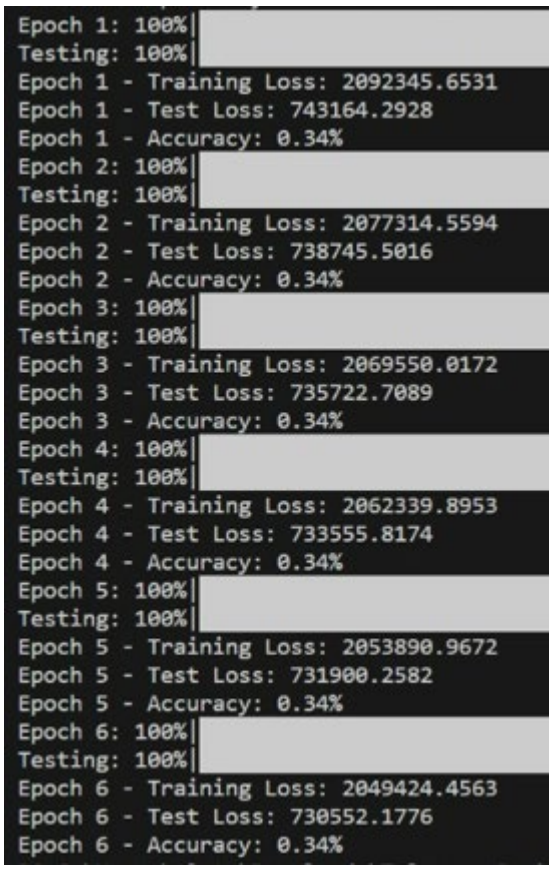
Додаток 12

Скриншот результатів роботи програми (1)



Додаток 13

Скриншот результатів роботи програми (2)



Додаток 14

Пес свійський, Текст 22

Текст 22

Цікаві факти

ДО

- За підрахунками Асоціації виробників товарів для домашніх тварин (США) майже 40 % американських родин тримають собак. Загалом у країні нараховується понад 78 мільйонів псів.
- Пекінез Паггі, десять років від роду, у 2011 потрапив до Книги рекордів Гіннеса як володар найдовшого язика на планеті серед собак. Його довжина становить 11,43 см.
- Данець Зевс із США зростом 1,12 м потрапив до Книги рекордів Гіннеса.
- Найбільшою собакою в світі є великий данець на ім'я Джордж. Зріст Джорджа становив 1,092 метра, повна довжина (від хвоста до носа) — 221 сантиметр, а вага — близько 110 кг. На рік такому псові потрібно більше 600 кг корму.
- Згідно з офіційним сайтом Книги рекордів Гіннеса титул найменшого собаки у світі вже котрий рік поспіль підтверджує крихітний чивава на прізвисько Бу-Бу (англ. Boo-Boo). Цей рекордсмен на зріст 10,16 см і вагою 675 гр.

ПІСЛЯ

- Асоціація виробників (-ць) товарів для домашніх тварин (США) підрахувала, що майже 40 % американських родин тримають собак. Загалом у країні нараховується понад 78 мільйонів псів.
- Пекінес Паггі в 10-річному віці у 2011 потрапив до Книги рекордів Гіннеса як володар найдовшого язика на планеті серед собак. Його довжина становить 11,43 см.
- Данець Зевс із США зростом 1,12 м потрапив до Книги рекордів Гіннеса.
- Найбільшою собакою в світі є великий данець на ім'я Джордж. Зріст Джорджа становив 1,092 метра. Повна довжина Джорджа (від хвоста до носа) становила 221 сантиметр. Вага Джорджа становила близько 110 кг. На рік такому псові потрібно більше 600 кг корму.
- Згідно з офіційним сайтом Книги рекордів Гіннеса, найменшим собакою у світі є чивава на прізвисько Бу-Бу (англ. Boo-Boo). Цей рекордсмен на зріст 10,16 см і вагою 675 гр.

Додаток 15

Пес свійський, Текст 16

Текст 16

Фізичні дані

ДО

Важливим критерієм оцінки собак для фахівців є їхня конституція — генетично обумовлений зв'язок властивостей і якостей тварини з особливостями статури і поведінки, що визначає її службові та племінні якості. З конституцією пов'язані здоров'я, життєстійкість, опірність, плодючість, тривалість життя і працездатність тварини.

Маса собаки залежить від його статури і кількості жирових запасів. Максимальний зареєстрований зріст у собаки — 109 см у холці при вазі 111 кг і довжині 221 см (порода — німецький дог). Англійський мастиф Зорба двічі потрапляв до Книги рекордів Гіннесса як найважчий собака у світі, в листопаді 1989 року його маса склала 343 фунти (155,6 кілограма) при висоті в холці 37 дюймів (94 сантиметри).

При регулярному тренуванні собаки на диво сильні для своїх розмірів — деякі здатні переносити на спині важку поклажу, тягти за собою сани й інші великі вантажі. Їздовим собакам часто доводиться ушістьóх тягти за собою нарти з масою одна тонна протягом кількох годин поспіль, фоксгаунди можуть іти по сліду 48 годин без перепочинку.

ПІСЛЯ

При оцінці собак фахівці (-чині) враховують конституцію собак. Конституція – це зв'язок властивостей тварини з особливостями статури та поведінки. Цей зв'язок визначає службові та племінні риси собаки. З конституцією пов'язані здоров'я, витривалість, плодючість, тривалість життя, працездатність тварини.

Маса собаки залежить від його статури і кількости жирових запасів. Рекордсменом у найвищому зрості є собака зростом 109 см при вазі 111 кг та довжині 221 см (німецький дог). Англійський мастиф Зорба двічі потрапляв до Книги рекордів Гіннеса як найважчий собака у світі. У листопаді 1989 року його маса була 343 унти (155,6 кілограма) при висоті в холці 37 дюймів (94 сантиметри).

При регулярних тренуваннях собаки можуть стати дуже сильними. Деякі здатні переносити на спині тяжкий вантаж, тягти за собою сани та інші великі вантажі. Їздові собаки часто вшістьох тягнуть за собою нарти масою одна тонна протягом кількох годин поспіль. Фоксгаунди можуть іти по сліду 48 годин без перепочинку.

Додаток 16

Пес свійський, Текст 12

Текст 12

Спеціальний курс дресирування: дресирування собак — провідників сліпих

ДО

Собака — помічник сліпого призначається для надання допомоги сліпому в орієнтуванні на місцевості і для заміни допомоги іншої (зрячої) людини. Для дресирування до цієї служби можуть використовуватися службові собаки різних порід: не дуже великі, але й не дрібні — від 50 до 68 сантиметрів зросту, урівноважені, спокійні, довірливі, з добре розвиненим зором і слухом.

Початкове дресирування собаки проводить дресирувальник, який повинен добре вивчити рухи сліпого в різних умовах, на різній місцевості і вміло наслідувати йому, використовуючи при цьому ціпок або звичайну палицю. Під час тренування собаки для допомоги сліпому потрібна також участь помічника. Таким помічником може бути громадський інструктор клубу службового собаківництва або хтось із членів сім'ї сліпого.

До дресирування за спеціальним курсом у собаки мають бути вироблені загальнодисциплінарні навички. До того ж собака повинен бути дуже слухняним.

До спеціальних навичок собак — провідників сліпих належать: рух попереду сліпого шляхом, вільним від перешкод, із зміною темпу руху; виконання поворотів у різних напрямках; зупинка перед перешкодами та

їхній обхід; підйом сходами, на природні підвищення і спуск із них; правильний рух (водіння сліпого) в умовах населених пунктів; водіння сліпого за певним маршрутом.

Для дресирування і використання собаки-провідника сліпого потрібна спеціальна шлейка. Вона складається з грудного ременя (ширина до 25 міліметрів, довжина до 80 сантиметрів) з пряжкою на кінці. До цього ременя за 18—20 сантиметрів від пряжки кріпиться поперечний ремінь завдовжки до 20 сантиметрів, що також має пряжку. Другий поперечний ремінь (завдовжки до 45 сантиметрів) кріпиться до грудного ременя за 24 сантиметра від першого. На кінці його є отвори для застібання.

У поперечний ремінь за 10 сантиметрів від грудного вшивають кільця для кріплення повідцевої дужки. Між кільцями нашивають обмежувачі з м'якої шкіри завдовжки 10 сантиметрів; між ними і грудним ременем залишають люфт у 5—8 сантиметрів. Обмежувачі одним кільцем щільно кріпляться до поперечного ременя, другим — на відстані 5—7 сантиметрів від нього.

Повідцеву дужку роблять із дроту (залізного, мідного тощо) діаметром 4—5 міліметрів і завдовжки до 100 сантиметрів. Стрижень дроту щільно обшивають м'якою шкірою, підбитою тканиною. До кінців дужки міцно пришивають пряжки для кріплення до кілець поперечних ременів.

ПІСЛЯ

Собака — помічник сліпого (-ої) потрібен для допомоги незрячій людині. Для цієї служби можуть використовуватися службові собаки різних порід. Вони мають бути не дуже великі, але й не дрібні (від 50 до 68 сантиметрів зросту), урівноважені, спокійні, довірливі, з добре розвиненими зором і слухом.

Початкове дресирування собаки проводить дресирувальник (-ця). Він (вона) має знати рухи сліпого (-ої) в різних ситуаціях і повторювати їх з ціпком. Під час тренування собаки для допомоги сліпому потрібна участь помічника (-ці). Ним (нею) може бути громадський (-а) інструктор (-ка) клубу службового собаківництва або хтось із членів сім'ї сліпого (-ої).

До дресирування за цим курсом у собаки мають бути вироблені загальні навички. Собака має бути дуже слухняним.

До спеціальних навичок собак — провідників сліпих належать: рух попереду сліпого (-ої) вільним шляхом із зміною швидкости руху; виконання поворотів у різних напрямках; зупинка перед перешкодами та їхній обхід; підйом сходами, на природні підвищення і спуск із них; правильний рух (водіння сліпого (-ої)) в умовах населених пунктів; водіння сліпого (-ої) за певним маршрутом.

Для дресирування і використання собаки-провідника сліпого (-ої) потрібна спеціальна шлейка. Вона складається з грудного ременя (ширина до 25 міліметрів, довжина до 80 сантиметрів) з пряжкою на кінці. До цього ременя за 18—20 сантиметрів від пряжки кріпиться поперечний ремінь завдовжки до 20 сантиметрів. Поперечний ремінь також має пряжку. Другий поперечний ремінь (завдовжки до 45 сантиметрів) кріпиться до грудного ременя за 24 сантиметра від першого. На його кінці є отвори для застібання.

У поперечний ремінь за 10 сантиметрів від грудного вшивають кільця для кріплення повідцевої дужки. Між кільцями нашивають обмежувачі з м'якої шкіри завдовжки 10 сантиметрів; між ними і грудним ременем залишають люфт у 5—8 сантиметрів. Обмежувачі одним кільцем щільно кріпляться до поперечного ременя, другим — на відстані 5—7 сантиметрів від нього.

Повідцеву дужку роблять із дроту (залізного, мідного тощо) діаметром 4—5 міліметрів і завдовжки до 100 сантиметрів. Стрижень дроту щільно обшивають м'якою шкірою, підбитою тканиною. До кінців дужки міцно пришивають пряжки для кріплення до кілець поперечних ременів.