Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Використання нейронних мереж для визначення схожості текстів українською мовою

Текстова частина до курсової роботи
за спеціальністю «Інженерія програмного забезпечення» - 121

Керівник курсової роботи
Кандидат технічних наук,
Старший викладач
Шабінська Марина Олегівна

_____

(підпис)
“____” _____ 2020 р.

Виконав студент ІПЗ-3:
Брус А. І.
“____” _____ 2020 р.

Київ 2020

Міністерство освіти і науки України

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»

Кафедра інформатики факультету інформатики

ЗАТВЕРДЖУЮ

Зав. кафедри інформатики,

к. ф.-м. н. С. С. Гороховський

_____

(підпис)

“____” _____ 2020 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

на курсову роботу

студента Бруса Андрія Ігоровича факультету інформатики 3 курсу

Тема: Використання нейронних мереж для визначення схожості текстів українською мовою

Зміст ТЧ до курсової роботи:

Індивідуальне завдання

Вступ

Розділ 1. Основні алгоритми та типи нейронних мереж для обробки природніх мов

Розділ 2. Розробка моделей

Розділ 3. Аналіз результатів

Висновки

Список використаної літератури

Дата видачі “____” _____ 2020 р. Керівник _____

(підпис)

Завдання отримав _____

(підпис)

Тема: Використання нейронних мереж для визначення схожості текстів українською мовою

Календарний план виконання роботи:

№	Назва етапу	Термін виконання	Примітка
1.	Отримання теми курсової роботи	25.10. 2019	
2.	Пошук технічної та наукової літератури за темою роботи	28.12. 2019	
3.	Ознайомлення з літературою	05.01. 2019	
4.	Огляд сучасних методів та найронних мереж для обробки природної мови	15.01. 2019	
5.	Ознайомлення з бібліотеками TensorFlow, Keras, gensim	07.02. 2020	
6.	Розробка моделей	02.03. 2020	
7.	Формування корпусу для тестування	29.03. 2020	
8.	Написання текстової частини роботи	14.04. 2020	
9.	Перегляд змісту роботи керівником	08.05. 2020	
10.	Внесення змін до курсової роботи відповідно до зауважень наукового керівника	09.05. 2020	
11.	Створення презентації	10.05. 2020	

Студент Брус А. І.

Керівник Шабінська М. О.

“____” _____

Зміст

Вступ

У сучасній інформатиці обробка природньої мови (NLP) – один із найбільш перспективних напрямів розвитку. NLP досліджує методи та проблеми аналізу та «розуміння» людської мови комп’ютерними системами.

Зазвичай у обробці природніх мов (NLP) виділяють три напрямки:

- Speech Recognition (Розпізнавання усної людської мови)
- Natural Language Generation (Генерація змістовних текстів)
- Natural Language Interpretation (Розуміння змісту, закладеного в текст)

Визначення семантичної подібності текстів належить до останньої категорії, NLI, оскільки розглядається саме значення тексту. Аналіз схожості текстів – складне завдання для комп’ютерних систем, оскільки схожість тісно пов’язана із семантикою.

У мовознавстві семантика – наука, що вивчає значення слів, тобто зміст, що вкладає людина у слово. Оскільки слова часто мають багато значень, що визначаються людиною інтуїтивно, ця проблема, як і обробка природніх мов (NLP) загалом, відноситься до класу AI-повних проблем, оскільки складність вирішення даної проблеми відповідає складності вирішення головного завдання штучного інтелекту — створення системи для обробки та аналізу інформації, еквівалентної людському мозку.

Актуальність теми. Кількість інформації, у тому числі і текстової, з кожним роком зростає все швидше. Виникає необхідність автоматизувати процеси аналізу та обробки цієї інформації, оскільки виконання такої роботи вручну є надзвичайно трудомістким завданням. Аналіз семантичної схожості текстів є одним з найбільш актуальних завдань NLP і корисний у таких випадках, як порівняння запитів до пошукових систем або служб підтримки для покращення запропонованих відповідей, визначення основної думки текстів, перевірки творів та статей на ідейний плагіат тощо. У зв’язку зі складністю обробки текстів,

написаних природними мовами, у галузі NLP широко використовуються методи машинного навчання, у тому числі і нейронні мережі.

Незважаючи на стрімко зростаючу кількість дослідницьких робіт у сфері машинного навчання та розробки нейронних мереж, кількість робіт про обробку української мови все ще є відносно малою. Більшість із них описують загальні методики вирішень проблем NLP або ж узагальнюють англомовний матеріал. Тому дослідження для української мови у цьому напрямку є доволі актуальними.

Мета і завдання дослідження. Метою даної роботи є аналіз існуючих нейронних мереж для NLI та розробка моделей, що можуть порівнювати два вхідних тексти, написані українською мовою.

Завдання дослідження:

1. Розглянути основні існуючі моделі представлення слів та речень у числовому вигляді;
2. Дослідити існуючі архітектури нейронних мереж для порівняння текстів;
3. Розглянути існуючі інструменти, розроблені для україномовних текстів;
4. На основі обраних алгоритмів реалізувати систему для порівняння україномовних текстів;
5. Протестувати систему та проаналізувати результати;

Наукове та практичне значення результатів. Теоретичні відомості та результати роботи можуть бути використані як основа для подальших досліджень у цьому напрямку. Розроблені моделі можуть використовуватись як для порівняння із іншими алгоритмами, так і для вдосконалення і створення максимально точної системи для порівнянь україномовних текстів.

Розділ 1. Основні алгоритми та типи нейронних мереж для обробки природніх мов

1.1 Рішення подібних проблем

Незважаючи на надзвичайну кількість подібних досліджень для англомовних текстів, порівняння текстів українською мовою недостатньо досліджено. Уваги заслуговують статті про методи аналізу україномовних текстів [1] та виявлення в них кореферентних пар [2]. Ці статті описують найбільш схожі проблеми, проте жоден з описаних там методів все ж не вирішує задачу, поставлену в даній роботі.

Із іншомовних статей найбільш значущими є [3], [4] та [5]. Усі вони англомовні та, за винятком [3], протестовані лише на англомовних корпусах текстів. Стаття [4] описує оновлену модель LSTM, яка, незважаючи на те, що відома уже відносно давно – із 1997 року [6], досі показує себе як одна з найефективніших при вирішенні задач NLP. У статті [5] описано структуру сіамських мереж для порівняння двох речень (довільної довжини).

Грунтовне дослідження багатьох методів на даних лікарських звітів проведено у статті [3], де найкраще себе показала архітектура LSTM та різні модифікації CNN (власне, дослідження було спрямоване саме на покращення CNN-алгоритмів). Завдання аналізу трохи відрізнялось, оскільки порівнювані тексти мали велику кількість подібних слів, а більш значущою для визначення схожості була наявність певних слів біля якихось спільних термінів (опис стану певних органів чи тканин, що описуються у кожному звіті). Проте, зважаючи на те, що форма звіту не є чітко структурованою, а самі звіти складаються людьми, основний механізм залишився незмінним – визначення спільних ознак в тексті та підрахунок міри їх схожості.

Зважаючи на результати, описані в цих статтях, для розробки експериментальних моделей порівняння текстів українською мовою були обрані структури LSTM та CNN.

1.2 Моделі представлення семантичних конструкцій (слів, речень та текстів)

Першою проблемою, із якою стикається NLI, є питання – як працювати із самими словами? Представлення слів як послідовності символів не відображає реальне смислове навантаження, яку несе слово. До прикладу, слова «парубок» та «хлопець» кардинально відрізняються, проте насправді є синонімами та означають одне і те ж; слова «гора» та «горе», навпаки, сильно відрізняються за значенням, незважаючи на відмінність лише в одному символі. Одним з найбільш вагомих кроків у розвитку NLI стало представлення послідовностей символів (як слів, так і цілих текстів) у вигляді векторів у певному просторі семантичних значень.

1.2.1 Bag of words

Мішок слів – спосіб представлення тексту у вигляді одного вектору, кожна позиція якого відповідає певному слову, а значення у цій позиції – кількості екземплярів цього слова, що зустрічаються у документі. Він часто використовується у сукупності з TF-IDF, що дають змогу оцінити частоту слова в тексті та розповсюдженість самого терміну, тобто кількість текстів, у яких зустрілось таке слово.

Недоліком мішка слів є те, що він абсолютно ігнорує порядок слів у тексті. Порівняння з його допомогою дають правильні результати лише при наявності однакових термінів у обох текстах, тоді як два речення, складені із синонімів, за цим алгоритмом не матимуть ніякого зв’язку. На базі даного методу працюють наївні баєсівські алгоритми, що в окремих задачах є доволі ефективними, проте в більшості випадків все ж програють у точності глибинним нейронним мережам.

1.2.2 Word2Vec

Word2Vec – спільна назва групи архітектур, представлена компанією Google (а, точніше, групою її дослідників під керівництвом Томаша Міколова) у 2013 році [7,8,9]. По суті Word2Vec є двошаровою нейронною мережею, що приймає на вхід текст та на виході формує множину нормалізованих векторів, де кожен вектор представляє певне слово із тексту. В основі цього підходу є твердження, що слово можна визначити, дивлячись на сусідні із ним слова, оскільки саме контекст визначає використання певного слова. Таким чином даний підхід дозволяє вловити деякі семантичні властивості слів, незважаючи на виключно статистичні підрахунки.

Спершу модель розроблялась для передбачення слів та генерації тексту, оскільки працює на основі статистичних показників, проте завдяки гарним результатам та своїй універсальності зараз основним її призначенням зазвичай є перетворення тексту із набору слів у формат, з яким можуть працювати більш складні глибинні нейронні мережі. Сама мережа має два основних алгоритми для перетворення слів у вектори: CBOW та skip-gram.

Неперервний мішок слів (CBOW) – передбачає слово, виходячи з контексту. Фактично, для кожного слова будується звичайний мішок слів розміру 2n+1, у який входять саме слово та n слів до та після нього. Порядок слів у самому мішку повністю ігнорується.

Skip-gram (словосполучення з пропусками) – навпаки, визначає контекст, виходячи із даного слова. У цьому підході враховується порядок слів. Значенням за замовчуванням у класичній реалізації word2vec є розмір вікна у n = 5 слів (5 попередніх відносно поточного слова та 5 наступних, тобто 11 у сумі); якщо ж слів у реченні менше, то вільні місця просто не враховуються.

Згідно з дослідженнями розробників [9], перший метод працює швидше, тоді як другий дає більш точні результати на великих корпусах даних. В обох випадках нейронна мережа підбирає коефіцієнти так, щоб косинусна міра подібності була якомога меншою між векторами слів, що часто зустрічаються в схожих

контекстах (тобто є велика ймовірність того, що вони подібні за значенням чи принаймні мають схожі ознаки).

Нехай w_1 та w_2 – два вектори, що є представленнями двох слів, тоді для них косинусна міра подібності P визначається таким чином:

$$P = \frac{w_1 * w_2}{\|w_1\|\|w_2\|}$$

Уваги також заслуговують n-грами – метод, при якому слово визначається за n кількістю слів, що зустрічаються в тексті до нього. Даний метод часто використовується для генерації текстів, адже дозволяє поступово, слово за словом, прогнозувати кожне наступне слово, достатньо лише знати ймовірності входження різних слів у даний текст. Проте для української мови його ефективність може бути нижчою, оскільки довільний порядок слів зменшує точність передбачень (у реченні слово, пов’язане з першим, може бути як другим, так і останнім в цьому реченні; при тому загальний сенс не зміниться). До того ж наступне слово в такому випадку визначається не лише попередніми словами.

1.2.3 Doc2vec

Doc2Vec – нейронна мережа, що працює за тим же принципом, що і word2vec, та призначена для векторного представлення більших структур, таких як речення, абзаци, або ж цілі тексти. Вона також має дві реалізації, що засновані на методах роботи word2vec.

PV-DM (Paragraph vector with Distributed Memory) – працює точно так само, як і CBOW. Єдиною відмінністю є те, що до векторів слів додається ще один, в якому закодовано ID документа, тобто вектор, що є унікальним для кожного документа (під документом мається на увазі будь-яка частина тексту: речення, параграф, чи весь документ). Схема роботи представлена на рисунку 1.

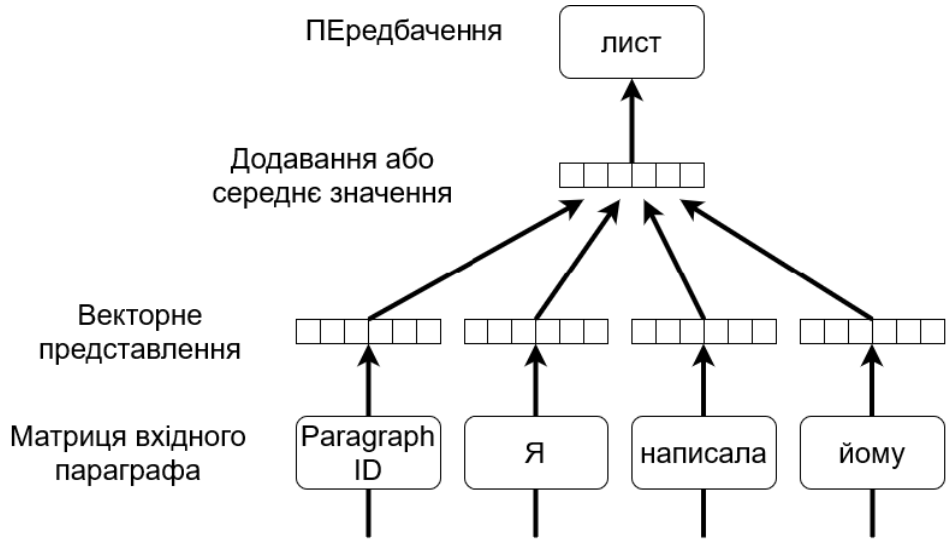

Рис. 1

PV-DBOW (Paragraph Vector with Distributed Bag Of Words) – алгоритм, заснований на skip-gram, та працює протилежно до PV-DM, як показано на рисунку 2:

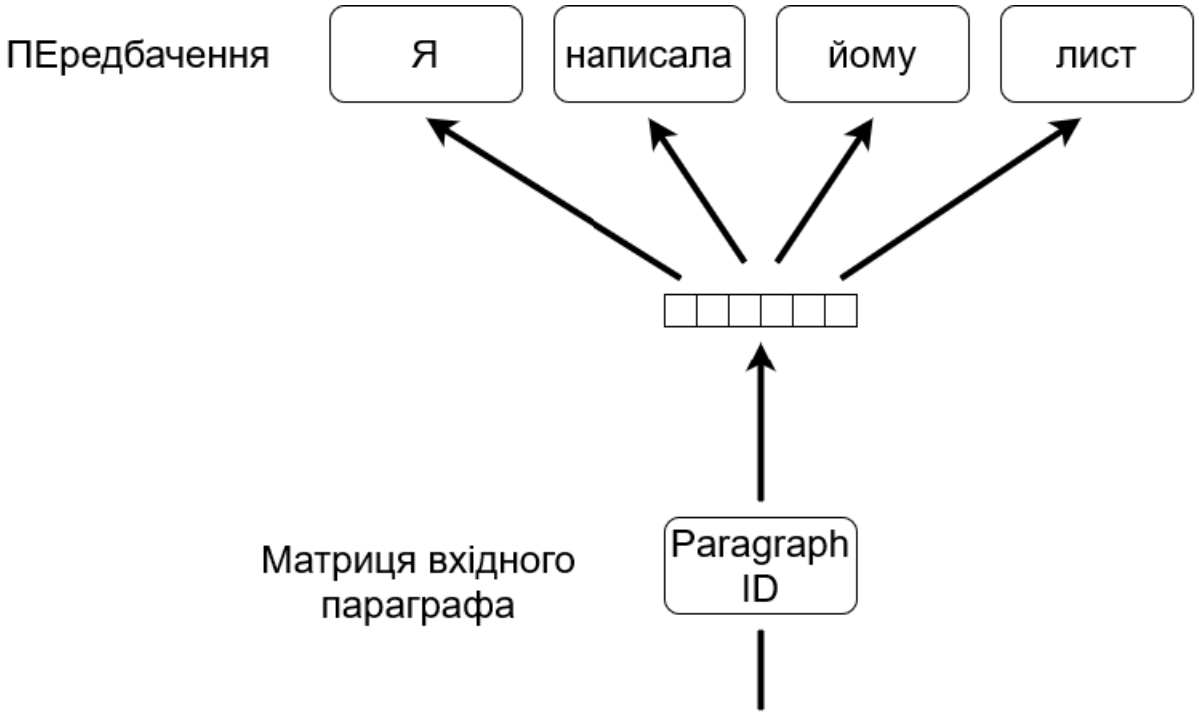

Рис. 2

Згідно з твердженням автора [11], PV-DBOW показує кращі результати та працює швидше, ніж PV-DM. Це пов’язано з тим, що, як показано на рисунку 5,

алгоритму PV-DBOW не зберігати та оброблювати вектори усіх слів при роботі. Автор також рекомендує використання не середнього значення, а суми векторів слів при обчисленні вектору документу [11].

Зважаючи на принцип роботи, doc2vec підходить для завдань порівняння текстів будь-якої довжини, оскільки зведення векторів (вектори слів у вектор речення, у вектор абзацу, декількох абзаців і т.д.) дозволяє обробляти тексти будь-якого розміру, аж до сотень тисяч слів. Проте, вони все ж показують гірші результати, ніж глибинні нейронні мережі (CNN та RNN) [3, 12].

1.3 Рекурентні та згорткові нейронні мережі

У задачах NLI найбільш ефективними себе показують глибинні нейронні мережі двох типів – рекурентні (RNN) та згорткові (CNN) [3, 13].

Успіх RNN пояснюється тим, що вони мають певний аналог пам’яті – прихований шар, параметр якого залежить від попереднього проходження цим шаром. Оскільки зміст найкраще визначається з контексту, тобто розташованих поряд слів, наявність пам’яті сильно впливає на якість результатів.

У CNN пам’ять відсутня, проте операція згортки при обробці речення дозволяє видобувати спільні ознаки із декількох сусідніх слів, а шар агрегації – виділяти найбільш важливі ознаки та на їх основі будувати семантичне представлення цілих послідовностей слів.

1.3.1 CNN

CNN (Згорткова нейронна мережа) - мережа, основними особливостями якої є використання шарів із операціями згортки (англ. – “convolution”) та агрегування (підвибірки, англ. – “pooling”). Структура CNN наслідує роботу мозкових клітин при роботі з органами зору, тож спочатку CNN зарекомендувала себе як одна з найкращих мереж для розпізнавання візуальних зображень. Проте завдяки особливостям векторного представлення слів операції згортки та агрегування також можуть бути використані над текстовими даними.

Принцип роботи CNN грунтується на теорії рецептивних полів. Даний підхід передбачає, що кожен нейрон обробляє не всю вхідну інформацію, а лише її частину, яка і називається рецептивним полем. Різні рецептивні поля частково перекриваються, що при об’єднанні усіх результатів дозволяє сформувати загальну картину.

Нейрон у згортковому шарі CNN отримує інформацію не від усіх нейронів попереднього шару, а лише від малої їх частини. Даний підхід успішно використовується для аналізу змісту тексту, оскільки подібний до роботи людського мозку, що визначає значення слова із сусідніх із ним слів (контексту),

а значення більших структур – із сукупності значень малих (слів та словосполучень).

Основою є операція згортки, що у NLP застосовується почергово до частин вхідної послідовності (або ж, простіше, до вікон із *n* слів).

Представимо вхідне речення як матрицю:

$$S \in R^{w \times l},$$

де *l* – довжина вхідного речення, *w* – довжина вектору, що представляє слово

Операція згортки використовує матрицю згортки $C \in R^{w \times n}$ до вікна розміром у *n* слів. Тоді, значення c_i для вікна $S[*, i, i+n]$ обраховуватиметься як:

$$c_i = \sigma\left(\sum (S[*, i, i+n]^{\circ} H) + b\right)$$

де *b* – параметр упередженості, ° - добуток Адамара,
σ – нелінійна активаційна функція (зазвичай сигмоїдна, тангенсна або ReLU)

Операція згортки застосовується до усіх можливих вікон у реченні (рисунок 3).

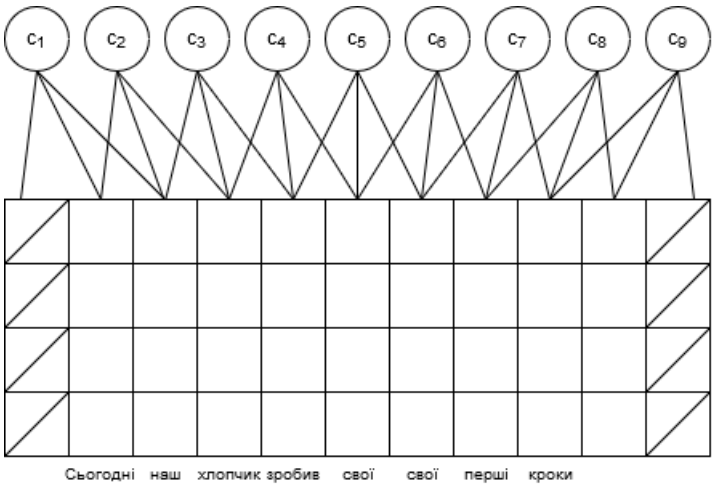

Рис. 3

Вихідним значенням буде карта ознак (або ж карта збуджень) *c*:

$$c = [c_1, c_2, \ldots, c_{l-n+1}],$$

$$\text{де c } \in R^{l-n+1}$$

Розміри матриці згортки фактично є параметрами ваг. Операція згортки є аналогом методу виділення n-грам у тексті, а карта ознак c – множина усіх можливих n-грам у цьому тексті (зважаючи на порядок слів). Згорткова мережа використовує матрицю однакової розмірності для усіх нейронів шару, тобто ваги та параметри упередженості теж є спільними. Це дозволяє мінімізувати кількість вільних параметрів, а також не допустити вибуху чи зникнення градієнту, що є частою проблемою нейронних мереж зі зворотним поширенням помилки (back propagation).

Іншим важливим шаром є агрегувальний (pooling), що застосовується для визначення найбільш важливих ознак та отримує на вхід карту ознак, побудовану згортковим шаром (Рисунок 4). Агрегувальний шар дозволяє зменшити розмірність оброблюваних даних, а разом з ними, відповідно, і кількість обчислень.

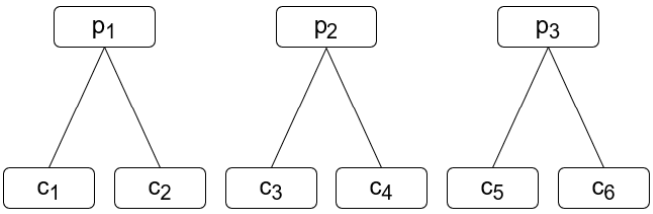

Рис. 4

Є декілька функцій для операції агрегації, із яких найбільш ефективною себе показує функція максимальної агрегації (max pooling) [12]. Для мінімалізації втрати даних пропонується брати фільтр розмірності 2. Тоді дві ознаки із карти ознак переводяться у одну результуючу ознаку таким чином:

$$p_i = \max\left(c_{2\times i-1}, c_{2\times i}\right)$$

Результатом проходження цього шару буде карта ознак:

$$p = \left[p_1, p_1, \dots, p_{\frac{l-n+1}{2}}\right],$$

де $p \in R^{\frac{l-n+1}{2}}$

Зменшення кількості обчислень дозволяє використовувати агрегатні шари для контролю перенавчання, а сама агрегація по суті підсумовує загальний сенс із послідовності векторних представлень частин тексту.

Для зведення усіх результатів останніми використовуються повноз'єднаний шар (fully connected layer) та шар втрат (loss layer). Кожен нейрон повноз'єднаного шару з'єднаний з усіма нейронами попереднього шару. Останній шар слугує для регуляції відхилень та втрат.

1.3.2 RNN

RNN (Рекурентна нейронна мережа) – частковий випадок рекурсивних нейронних мереж, що працюють «у часі». RNN використовують рекурентний прихований стан, який залежить від попереднього проходження (кроку), що по суті є аналогом короткочасної пам'яті. Завдяки рекурентній структурі RNN здатні працювати з вхідними послідовностями даних довільної довжини, що робить їх ефективними при обробці текстів та NLI.

Розгорнута структура RNN зазвичай має вигляд, зображений на рисунку 5:

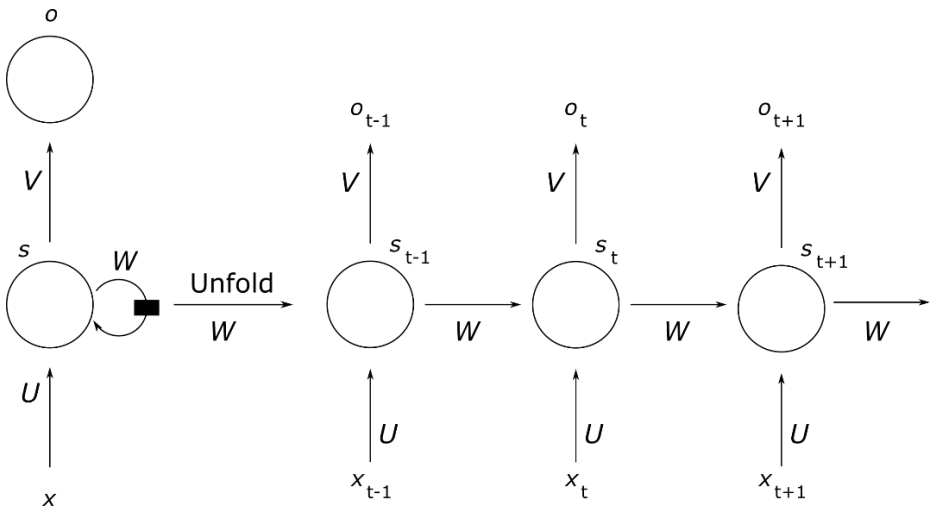

Рис. 5

Отримуючи на вхід послідовність (речення) із n слів, розгорнута RNN матиме вигляд n-шарової мережі, де кожен шар відповідатиме за одне слово. Таким чином, роботу RNN можна описати так:

- x_t – вхідні дані на кроці t (у даному випадку, x_i – векторне представлення i-го слова в реченні);
- o_t – вихідні дані на кроці t.
- U – матриця ваг для вхідних даних поточного стану;
- W – матриця ваг із попереднього стану;
- σ – функція активації;
- s_t – прихований стан на кроці t, або ж «пам'ять» мережі на цьому кроці. Він залежить від попереднього стану та вхідних даних на поточному стані:

$$s_t = \sigma(Ux_t + Ws_{t-1})$$

Зазвичай вводять також нульовий вектор s_0 для обрахування першого прихованого кроку.

Для контролю похибки (та, власне, тренування) використовується зворотне поширення у часі (BPTT) – модифікація стандартного алгоритму зворотного поширення, що працює на основі методу схоластичного градієнтного спуску. BPTT рухається не назад (як back propagation), а вздовж розгорнутої у часі мережі, та вираховує зміну ваг, виходячи з похідної похибки відносно цієї ваги.

Проте градієнтний спуск такого виду має значні проблеми із вибухом та зникненням градієнту, особливо на великих об'ємах вхідних даних, що ускладнює видобування довготривалих залежностей (long-term dependencies). Це знижує її ефективність при семантичному чи морфологічному аналізі мовлення та визначенні загальних тональностей тексту.

1.3.3 LSTM

У 1997 році була представлена архітектура LSTM (Long Short-Term Memory) – модифікація RNN, що використовує так звані забувальні ворота або ж фільтри (forget gates) та вводить спеціальну комірку пам'яті c_t. Таким чином, пам'ять про

попередній стан може зберігатись впродовж не одного кроку, а необмежену кількість часу (аж до сотень і тисяч кроків). Такий підхід дає змогу виявляти довготривалі залежності та усуває проблему вибуху та зникання градієнту [6].

Формально принцип роботи LSTM можна описати так:

- Вхідний фільтр i_t контролює те, наскільки важливі на даному кроці вхідні дані x_t:

$$i_t = \sigma(U_i x_t + W_i s_{t-1} + b_i)$$

- Забувальний фільтр f_t визначає, чи варто нехтувати даними попереднього кроку:

$$f_t = \sigma(U_f x_t + W_f s_{t-1} + b_f)$$

- Тангенсна функція τ_t використовується як функція активації:

$$\tau_t = \tan(U_\tau x_t + W_\tau s_{t-1} + b_\tau)$$

- Значення самої комірки пам’яті вираховується як:

$$c_t = i_t \circ \tau_t + f_t \circ c_{t-1}$$

- Вихідний фільтр o_t визначає, яка частина c_t перейде до іншого кроку:

$$o_t = \sigma(U_o x_t + W_o s_{t-1} + b_o)$$

- Значення стану вираховується як:

$$s_t = o_t \circ \tan(c_t)$$

Де ° - добуток Адамара (посимвольне множення);
σ – логістична функція активації воріт $\sigma(x) = (1 + e^{-x})^{-1}$;
b – параметр упередження; W, U – матриці ваг мережі.

Нульові s_0 та c_0 ініціалізуються для проходження першого кроку.

Дана архітектура показує гарні результати на виявленні змістових зв’язків між далекими частинами тексту. Наприклад, зв’язок між першим та останнім словом

у реченні, що часто трапляється у мовах слов'янської групи з непрямим (варіативним) порядком викладення слів.

1.3.4 Сіамські нейронні мережі (Siamese neural networks)

Сіамська архітектура нейронної мережі давно відома та широко використовується у задачах для порівнянь двох наборів даних [14]. Її суть полягає у використанні двох повністю ідентичних нейронних мереж, які поєднує спільний останній шар (інколи декілька останніх).

На вхід при тренуванні мережа приймає три значення (w_1, w_2, s), де w_1 та w_2 – векторні представлення тексту, $s \in \{0,1\}$ – очікуваний показник схожості між w_1 та w_2 (1 – схожі, 0 – ні). Кожній підмережі передається один вектор, обидва вектори проходять одну і ту ж обробку (кількість шарів у мережах, їх вид та параметри повністю співпадають).

Останній, спільний шар, отримує на вхід дві результуючі ознаки та визначає їх схожість, вимірюючи Манхеттенську відстань між двома ознаками [4, 5]:

$$\exp(-\|v_1 - v_2\|) \in [0,1] ,$$

де v_1 та v_2 – ознаки, що отримуються на виході з підмереж.

Метою тренування є якомога менша відстань для схожих та якомога більша для несхожих об'єктів.

1.4 Особливості NLI та огляд існуючих технологій для української мови

Наразі переважна більшість алгоритмів та їх реалізацій створена для англійської мови. Проте українська мова за структурою значно відрізняється. Особливо обробку тексту ускладнює непрямий порядок слів у реченні (присудок не завжди йде після підмета, а може бути у зовсім іншому кінці речення, як перед підметом, так і після нього), а також велика кількість форм слова (відмінки, множини тощо). Друга проблема вирішується шляхом лематизації вхідних даних на етапі

попередньої обробки тексту, проте перша є більш складною та може потребувати змін у структурі моделей та більш глибоких досліджень.

Кількість існуючих технологій для NLP, спеціалізованих для української мови, є доволі малою. JLanguageTool [15] – застосунок, що має мобільну, веб та десктоп-версії, а також Java API, для обробки більшості європейських мов. Може визначати орфографічні та граматичні помилки у тексті, а також розмічувати текст (визначати для слів частини мови та інші ознаки).

Згідно зі статистикою сайту (станом на квітень 2020 року), українська є однією з найбільш використовуваних мов, а саме API має багато практичних застосувань (як-от оцінка загальної грамотності тексту, або ж визначення кореферентних зв’язків). Проте зважаючи на специфіку обраної теми, такі підходи не несуть практичної цінності, а її використання у даній роботі обмежується використанням утиліт для лематизації та токенізації тексту [10].

Іншим корисним ресурсом є lang.org.ua [16] – відкрита спільнота, що займається збором текстів українською, формуванням корпусів текстів та векторних моделей, а також розмічених текстів та інших даних для NLP. Особливу користь для даної роботи представляють розроблені цією спільнотою моделі векторних представлень українських слів та тестові набори, аналогічні поширеним англомовним наборам.

Розділ 2. Розробка моделей

На жаль, оскільки тематика порівнянь україномовних текстів раніше грунтовно не розглядалась, розмічені тренувальні корпуси для таких цілей також відсутні. Електронний ресурс lang-ua містить велику кількість текстів, проте через ліцензійні обмеження більшість із них знаходяться у неструктурованому вигляді (тобто речення у них перемішані та розташовані у довільному стані). Цілісність текстів збережена у корпусі судових актів, але така тематика є дуже специфічною та обмеженою в лексиконі.

У зв'язку з відсутністю розмічених даних виникла потреба створити необхідний корпус власноруч. Оскільки порівняння текстів людиною є доволі складним та трудомістким завданням, було прийнято рішення проаналізувати і розмітити вручну корпус речень, та реалізувати моделі для роботи з реченнями в якості коротких текстів. Водночас для кожної моделі запропоновано метод, який дозволяє легко модифікувати її для роботи з більшими текстами.

Для розробки моделей було обрано дві архітектури: CNN та LSTM, оскільки вони показують найкращі результати при вирішенні таких проблем [3, 12, 4].

Метод Doc2Vec не було розглянуто на практиці, оскільки для порівняння речень зручніше використовувати звичайну суму векторних представлень слів [8]. Однак, doc2vec пропонується як доповнення до моделі CNN. Для порівняння цілих текстів за допомогою CNN для кожного речення або параграфа застосувати doc2vec, та представити текст як множину векторних представлень речень (параграфів). Це дозволить аналізувати цілі тексти, майже не змінюючи структуру самої CNN.

В той же час, CNN показує кращі результати при визначенні ознак для словосполучень, часто вживаних фраз та локальних семантичних зв'язків. Це пояснюється структурою мережі, а саме матрицями операцій згортки, що застосовуються лише до сусідніх елементів. Тому CNN, скоріш за все, покаже гірші результати при роботі з великими текстами.

У випадку LSTM дані також можна подавати не у вигляді послідовності векторних представлень слів, а представлень цілих речень або абзаців. Однак, архітектура цієї мережі дозволяє приймати на вхід послідовності довільної довжини, тож можливим рішенням також є подача усього тексту цілком, без поділу на речення, у вигляді послідовності векторів слів. Проте ці припущення також потребують додаткових досліджень на розмічених корпусах даних.

Варто згадати про метод LSA, що також показує значні результати, оскільки розроблений спеціально для семантичного аналізу текстів. Проте він є звичайним алгоритмом, що не використовує нейронні мережі та машинне навчання, тому у цій роботі він не розглядається.

2.1 Попередня обробка текстів

Перед безпосереднім аналізом кожен текст проходить попередню обробку. Вона включає в себе приведення до нижнього регістру усіх слів, токенізацію та лематизацію. Для токенізації та лематизації використовуються скрипти LanguageTool API ua [10].

Для навчання обох нейронних мереж використовується готова модель word2vec, представлена на порталі lang-ua [16]; розмірність векторів n = 300. Проведені експерименти для двох моделей: звичайної (lowercase model) та навченої на лематизованих текстах (lowercase lemmatized model). Відповідно, лематизація тестових даних проводилась лише для другої моделі.

Загальний аналіз проводився на власноруч розміченому корпусі, що містить пари випадково обраних речень із корпусів «Новини» та «Художня література», представлених на порталі lang-ua [16]. Розмір корпусу становить 1200 пар.

Для кожної пари речень обиралось число 0 або 1, де 0 означає, що речення несхожі, а 1 – схожі. Схожість визначалась за наступними правилами:

1. Речення схожі, якщо описують одне і те ж явище словами-синонімами: «Щодня я іду цим шляхом» – «Кожного дня я крокую цією дорогою»

2. Речення схожі, якщо одне з них містить значно більше слів, проте є поширеною версією коротшого речення (з урахуванням правила номер 1): «Щодня я іду цим шляхом» – «Кожного божого дня із радістю в серці я крокую цією дорогою»
3. Речення схожі, якщо описують одне явище, але одне з них заперечує інше: «Я завжди іду цим шляхом» – «Я ніколи не іду цим шляхом»
4. У зв'язку з випадковим вибором речень переважають несхожі пари, тому у спірних випадках (для речень, де складно визначити, чи схожі вони) обиралось число 1.
5. У інших випадках пари речень позначались як несхожі.

Розмір тренувальної вибірки становить 70% від загального розміру корпусу, розмір валідаційної вибірки – 30%.

2.2 Підготовка векторних моделей

Оскільки кількість слів у мові дуже велика, майже неможливо представити у векторній моделі усі слова в мові. Тому у тренувальних даних можуть зустрічатися слова, що не представлені у векторній моделі. Найефективнішим вирішенням цієї проблеми є повторне тренування векторної моделі на тому ж корпусі даних, з яким буде проводитись подальша робота. Проте для невеликих тестових корпусів цей метод не підходить, оскільки результати виходять занадто упередженими через малу вибірку даних.

Іншим шляхом вирішення цієї проблеми є ініціалізація випадковими векторами тих слів, яких немає у моделі. Цей підхід і було обрано у даній роботі.

Спершу, для пришвидшення обробки, кожне слово тестового сету переводиться у числове представлення (за допомогою звичайного словника та id слова у цьому словнику). Далі матриця ваг для Embedding-шару ініціалізується випадковими числами за нормальним розподілом:

```
# розміри матриці - (к-сть слів)x(розмірність вектора слова)
emb_matrix = 1 * numpy.random.randn(len(all_words) + 1, dim)
```

Потім існуючим у векторній моделі словам виставляються відповідні числа:

```
for word, index in all_words.items():
  if word in w2v_model.vocab:
    emb_matrix[index] = w2v_model.word_vec(word)
```

2.3 модель LSTM

Дана модель працює на основі Siamese Manhattan LSTM, тобто використовує функцію манхеттенської відстані між результатами двох нейронних підмереж для визначення схожості [4].

Приклад власної реалізації шару, що обраховує манхеттенську відстань (розширення класу Layer):

```
class ManhattanLayer(Layer):
  def __init__(self, **kwargs):
    self.result = None
    super(ManhattanLayer, self).__init__(**kwargs)

  # викликається автоматично, вхідні шари компілюються тут
  def build(self, input_shape):
    super(ManhattanLayer, self).build(input_shape)

  # виконує всю логіку шару
  def call(self, inputs, **kwargs):
    self.result = K.exp(-K.sum(K.abs(inputs[0] - inputs[1]), axis=1,
      keepdims=True))
    return self.result

  def compute_output_shape(self, input_shape):
    return K.int_shape(self.result)
```

Максимальна довжина вхідної послідовності (тексту) – 24 слова. Варто зазначити, що в архітектурі RNN, на відміну від CNN, розмір обмежений лише системними ресурсами, тож число обране з огляду на те, що порівнюються речення. Збільшення розміру дозволить передавати на вхід одразу цілі параграфи тексту довільної довжини. Розмір у 24 слова обрано емпірично (згідно із [17], середня довжина українських слів складає 13 слів, середньо-квадратичне відхилення – 4.38 слова, тож розміру у 24 терміни має бути більш ніж достатньо, проте цей аспект також потребує додаткових досліджень).

Початковий шар – шар представлень (Embedding), що ініціалізується готовими моделями формату word2vec (розмірність d = 300). Надалі ваги блокуються, тобто при подальшому навчанні значення цього шару не змінюються:

```
sub_model = Sequential()
sub_model.add(Embedding(len(w2v_matrix), w2v_dim, weights=[w2v_matrix],
                        input_shape=(max_sent_size,), trainable=False))
```

Кількість прихованих шарів у кожній підмоделі – 50. Дві підмоделі об'єднує спільний шар, що отримує на вхід два вектори та продукує остаточне значення (функція активації – manhattan distance):

```
input1 = Input(shape=(max_sent_size,), dtype='int32')
input2 = Input(shape=(max_sent_size,), dtype='int32')

shared_layer = tools.ManhattanLayer()([sub_model(input1),
                                       sub_model(input2)])
```

У якості функції втрат була обрана середньоквадратична похибка - mse (mean squared error):

```
model = Model(inputs=[input1, input2], outputs=[shared_layer])
model.compile(loss='mean_squared_error',
              optimizer=Adam(), metrics=['accuracy'])
```

Метрика, використана для оцінки моделі – accuracy, тобто кількість вірно передбачених результатів відносно усіх передбачень. Іншими словами, якщо P – кількість вірно прийнятих рішень, а N – загальна кількість порівнюваних пар, то:

$$accuracy = \frac{P}{N}.$$

Відповідно, результат тренування оцінюється за accuracy для тестового сету.

2.4 модель CNN

Модель CNN також реалізована з використанням сіамської структури та манхеттенською функцією активації (реалізація така ж, як і для моделі попереднього розділу). За основу кожної підмережі взято структуру, запропоновану у [18]. Загальну архітектуру можна побачити на рис.6:

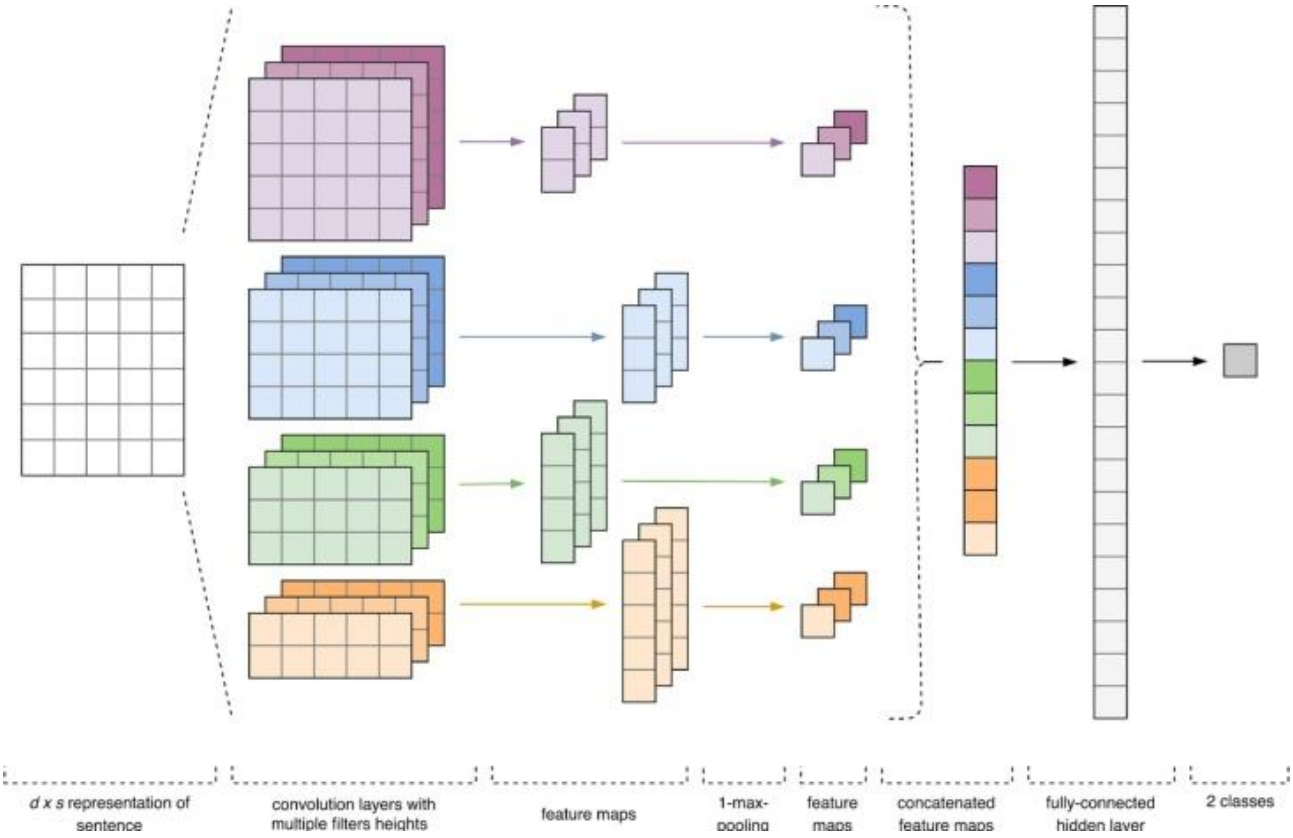

Рис. 6 [19, 18]

Вхідні шари та Embedding-шар такі ж, як і в попередній моделі. У наступних шарах використовуються згорткові матриці чотирьох розмірностей m = (2, 3, 4, 5). Це дозволяє мережі вловити семантичні значення дво-, три-, чотири- та п’яти-грам у тексті, що підвищує точність обчислень [18].

Додано 10 паралельних шарів для кожної розмірності, та така ж кількість шарів агрегації.

Для функції активації було обрано ReLU. У якості шару втрат використано шар

dropout-регуляризації з імовірністю оновлення p = 0.3. Останнім було додано повнозв'язний шар розмірністю у 30 нейронів.

Функція, що описує основну частину підмоделі:

```
def build_cnn_model(input_layer, w2v_matrix):
  sub_model = Embedding(len(w2v_matrix), w2v_dim, weights=[w2v_matrix],
              input_shape=(max_sent_size,), trainable=False)(input_layer)
  sub_model = Dropout(0.2)(sub_model)
  convs = []
  for window_size in [5, 4, 3, 2]:
    for i in range(10):
      x = Conv1D(filters=1, kernel_size=window_size, padding='valid',
          activation='relu')(sub_model)
      x = GlobalMaxPool1D()(x)
      convs.append(x)
  sub_model = concatenate(convs, axis=1)
  sub_model = Dropout(0.3)(sub_model)
  sub_model = Dense(30, activation='relu')(sub_model)
  return sub_model
```

Обидві підмережі об'єднує останній шар, що приймає остаточне рішення, використовуючи манхеттенську функцію для активації.

```
shared_layer = tools.ManhattanLayer()([sub_model1, sub_model2])
model = Model(inputs=[input1, input2], outputs=[shared_layer])
model.compile(loss='binary_crossentropy', optimizer=Adam(),
  metrics=['accuracy'])
```

Функція втрат – бінарна кросс-ентропійна функція, метрика оцінки – *accuracy*.

2.5 Застосовані технології та корпуси даних

Для розробки моделей було обрано мову програмування Python. При обробці векторних моделей було використано бібліотеку gensim, оскільки вона є найбільш оптимізованою для роботи з великими колекціями даних (завдяки багатопотоковій обробці, пострічковому завантаженні інформації для обробки та іншим алгоритмам оптимізації обчислень).

Для розробки моделей було використано бібліотеку Tensorflow та Keras, що входить до її складу. Перша є відкритою бібліотекою для машинного навчання,

розроблена та підтримується компанією Google. Друга спеціалізується саме на глибинних нейронних мережах, розроблена інженерами-ентузіастами, та згодом включена до складу Tensorflow. Keras пропонує засоби для роботи зі згортковими та рекурсивними нейронними мережами, а також містить вбудовані реалізації структур шарів (вхідних, вихідних, агрегатних та багатьох інших) та різноманітні активаційні функції.

Для роботи використовувались векторні моделі lowercase word2vec та lemmatized lowercase word2vec, що розміщені у вільному доступі на порталі lang-ua [16]. Для токенізації та лематизації текстів при попередній обробці використовується LanguageTool API uk для Python [10]. Список стоп-слів було сформовано власноруч. Основні датасети також було розмічено вручну, також на основі корпусу текстів з порталу lang-ua.

Розділ 3. Аналіз результатів

Тестування обох моделей проводилось для двох випадків: лематизованих речень та не лематизованих речень, а також відповідних векторних моделей.

Для досягнення максимальної ефективності навчання проводилось у багато епох – по 100 для кожної моделі. Із кожного заходу була обрана найбільш результативна епоха (модуль с. Далі для неї знімалось блокування шару векторних представлень, та проводилось додаткове тренування протягом 20 епох. Внаслідок такого підходу виявлено доволі значний приріст точності.

Результати для нелематизованих речень та векторної моделі продемонстровано у таблиці 1.

Модель	*accuracy*
CNN	0.6061
CNN з розблокованим embedding-шаром	0.6132
LSTM	0.6057
LSTM з розблокованим embedding-шаром	**0.6142**

Таблиця 1.

Результати для лематизованих речень та векторної моделі продемонстровано у таблиці 2.

Модель	*accuracy*
CNN	0.6299
CNN з розблокованим embedding-шаром	0.6386
LSTM	0.6368
LSTM з розблокованим embedding-шаром	**0.6401**

Таблиця 2.

Із обох таблиць видно, що знаходження найкращої епохи та подальше тренування на її основі з розблокованим шаром представлення дає змогу досягнути кращих результатів.

Для лематизованих речень до розблокування шару значно кращих результатів досягла LSTM, тоді як для нелематизованих – CNN, проте із дуже незначною перевагою. В загальному, краще себе показує LSTM, але це може бути частковим випадком та пов’язаним із вибіркою корпусу. Для більшої достовірності результатів все ж необхідно додаткове тестування на інших корпусах розмічених даних, яких на час написання статті ще не створено.

Низькі загальні показники перш за все пояснюються малою розмірністю навчального сету, адже в невеликих сетах різниця між тренувальною та тестовою вибіркою є більш відчутною. При збільшенні корпусу відмінності згладжуються, що дозволить потенційно отримати значно кращі результати.

Результати також можуть бути покращені шляхом зміни активаційних функцій та кількостей різних шарів. Існує велика кількість модифікацій, які можуть бути застосовані до обраних архітектур для кращих результатів.

На жаль, тема семантичної схожості текстів саме для української мови є мало дослідженою. Інші архітектури, такі як LSA, LDA, Doc2vec, WMD, потенційно можуть показати кращі результати. Це також потребує значних додаткових досліджень та модифікацій існуючих алгоритмів.

Висновки

У цій роботі проведено дослідження основних нейронних мереж для аналізу подібності україномовних текстів та на практиці розглянуто використання нейронних мереж для порівняння текстів українською мовою. Досліджено основні типи нейронних мереж для порівняння текстів, обробки і векторного представлення слів та текстів. Також розглянуто існуючі інструменти, розроблені для україномовних текстів, та перспективи їх використання для подібних задач.

Детально описано та реалізовано дві моделі, CNN та LSTM, для порівняння текстових даних у вигляді речень. Розроблені моделі можуть бути використані як для порівняння та базової реалізації при розробці інших моделей, так і для покращення результатів власне цих моделей. Також досліджено вплив попередньої лематизації текстів. Загалом результати свідчать, про те, що моделі ще потребують вдосконалення, а також тренування на більших корпусах даних.

Скорочення та абревіатури

NLP – Natural Language Processing (обробка природної мови), напрям на перетині інформатики та лінгвістики, що досліджує проблеми та методологію синтезу, аналізу та розпізнавання людської (природної) мови, а також роботу з великими об’ємами лінгвістичних даних.

NLI (NLU) – Natural Language Interpretation (Natural Language Understanding), напрям NLP, що спеціалізується на аналізі, категоризації та порівнянні лінгвістичних даних.

TF-IDF (term frequency - inverse document frequency) – статистичний показник, що означає важливість певного терміна для документу у певному корпусі документів. Пропорційний кількості екземплярів цього слова у тексті та обернено пропорційний до кількості текстів у корпусі, що містять таке слово.

RNN – Recurrent Neural Network, глибинна нейронна мережа, окремий випадок рекурсивних нейронних мереж, у якій замість декількох послідовних шарів дані проходять рекурсивно через один і той же шар, при чому кожен наступний враховує пов'язаний з даними з попереднього виклику.

CNN – Convolutional Neural Network, глибинна нейронна мережа, особливістю якої є використання шарів згортки та агрегації (останній необов’язковий).

LSTM – Long Short-Time Memory, розширення архітектури RNN, що покликана усунути проблему вибуху та зникання інградієнту при обробці великих наборів даних.

CBOW – Continuous Bag Of Words, окремий випадок мішка слів, один із методів обчислень, що використовуються у векторних моделях представлення слів.

LSA – Latent Semantic Analisator – алгоритм, розроблений у 1988 році для семантичного аналізу слів. Не використовує принципів машинного навчання

Список використаної літератури

1 Метод виявлення кореферентних пар в україномовному тексті з використанням згорткових нейронних мереж. Погорілий С. Д., Крамов А. А. [Електронний ресурс] – 2019. Режим доступу до ресурсу: https://visnit.uu.edu.ua/upload/publicationpdf/e6d2d7e79f4052b9061e747e2a7dbfc0.pdf

2 Метод аналізу когерентності україномовних текстів із використанням рекурентної нейронної мережі. Погорілий С. Д., Крамов А. А., Яценко Ф. М. [Електронний ресурс] – 2019. Режим доступу до ресурсу: http://www.immsp.kiev.ua/publications/articles/2019/2019_4/04_Pogorelyi_19.pdf

3 Detection of medical text semantic similarity based on convolutional neural network. Tao Zheng, Yimei Gao, Fei Wang, Chenhao Fan, Xingzhi Fu, Mei Li, Ya Zhang, Shaodian Zhang and Handong Ma. [Електронний ресурс] – 2019. Режим доступу до ресурсу: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6686478/

4 Siamese Recurrent Architectures for Learning Sentence Similarity. Aditya Thyagarajan, Jonas Mueller. [Електронний ресурс] – 2018. Режим доступу до ресурсу: https://www.researchgate.net/publication/307558687_Siamese_Recurrent_Architectures_for_Learning_Sentence_Similarity

5 Learning Text Similarity with Siamese Recurrent Networks. Paul Neculoiu, Maarten VersteeghandMihai Rotaru. [Електронний ресурс] – 2016. Режим доступу до ресурсу: https://www.aclweb.org/anthology/W16-1617.pdf

6 Long Short-Term Memory. Sepp Hochreiter and Jürgen Schmidhuber. [Електронний ресурс] – 1997. Режим доступу до ресурсу: https://www.mitpressjournals.org/doi/abs/10.1162/neco.1997.9.8.1735

7 Linguistic Regularities in Continuous Space Word Representations. Tomas Mikolov, Wen-tau Yih, Geoffrey Zweig. [Електронний ресурс] – 2013. Режим доступу до ресурсу: https://www.aclweb.org/anthology/N13-1090.pdf

8 Distributed Representations of Words and Phrases and their Compositionality. Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, Jeffrey Dean. [Електронний ресурс] – 2014. Режим доступу до ресурсу: https://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf

9 Word2vec, Tomas Mikolov, 2013. [Електронний ресурс] – Режим доступу до ресурсу: https://code.google.com/archive/p/word2vec/

10 LanguageTool API NLP uk. Андрій Рисін. [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/brown-uk/nlp_uk (Дата звернення - 27 березня 2020)

11 Distributed Representations of Sentences and Documents. Quoc V. Le, Tomas Mikolov. [Електронний ресурс] – 2014. Режим доступу до ресурсу: https://arxiv.org/abs/1405.4053

12 Multi-Perspective Sentence Similarity Modeling with Convolutional Neural Networks. Hua He, Kevin Gimpel, and Jimmy Lin. [Електронний ресурс] – 2015. Режим доступу до ресурсу: https://www.aclweb.org/anthology/D15-1181.pdf

13 Combination of Convolutional and Recurrent Neural Network for Sentiment Analysis of Short Texts. Xingyou Wang, Weijie Jiang, Zhiyong Luo. [Електронний ресурс] – 2016. Режим доступу до ресурсу: https://www.aclweb.org/anthology/C16-1229.pdf

14 Signature Verification using a "Siamese" Time Delay Neural Network. Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Sickinger and Roopak Shah. [Електронний ресурс] – 1994. Режим доступу до ресурсу: http://papers.nips.cc/paper/769-signature-verification-using-a-siamese-time-delay-neural-network.pdf

15 LanguageTool.Org. [Електронний ресурс] – Режим доступу до ресурсу: https://www.languagetool.org/ (Дата звернення – 12 квітня 2020)

16 Lang-uk. [Електронний ресурс] – Режим доступу до ресурсу: https://lang.org.ua/uk/ (Дата звернення – 22 березня 2020)

17 Статистичний розподіл і флуктуації довжин речень в українських, російських і англійських корпусах. О. С. Кушнір, О. С. Брик, В. Є. Дзіковський, Л. Б. Іваніцький, І. М. Катеринчук, Я. П. Кісь. [Електронний ресурс] – 2016. Режим доступу до ресурсу: http://science.lpnu.ua/sites/default/files/journal-paper/2018/jun/12983/22228-239.pdf

18 A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. Ye Zhang, Byron Wallace. [Електронний ресурс] – 2015. Режим доступу до ресурсу: https://arxiv.org/abs/1510.03820

19 Convolutional Neural Network for sentiment analysis of text. [Електронний ресурс] (Дата звернення: 5 квітня 2020). Режим доступу до ресурсу: https://habr.com/ru/company/mailru/blog/417767/