Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

магістр
(назва освітнього ступеня)

на тему: *Методи автоматизованого перекладу природної мови на основі нейромережевої моделі "послідовність-послідовінсть"*

Виконав(ла): студент(ка) *6* курсу, групи *СІм-61*
спеціальності *123 Комп'ютерна інженерія*
(шифр і назва спеціальності)

_____ *Луцишин Р. О.*
(підпис) (прізвище та ініціали)

Керівник _____ *Луцків А. М.*
(підпис) (прізвище та ініціали)

Нормоконтроль _____ *Луцик Н. С.*
(підпис) (прізвище та ініціали)

Завідувач кафедри _____ *Осухівська Г. М.*
(підпис) (прізвище та ініціали)

Рецензент _____ *Баран І. О.*
(підпис) (прізвище та ініціали)

Тернопіль
2020

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет Факультет комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)

Кафедра Кафедра комп'ютерних систем та мереж
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ *Осухівська Г. М.*
(підпис) (прізвище та ініціали)
« » 2020 р.

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня *магістр*
(назва освітнього ступеня)

за спеціальністю *123 Комп'ютерна інженерія*
(шифр і назва спеціальності)

студенту *Луцишину Роману Олеговичу*
(прізвище, ім'я, по батькові)

1. Тема роботи *Методи автоматизованого перекладу природної мови на основі нейромережевої моделі "послідовність-послідовність"*

Керівник роботи *Луцків Андрій Мирославович, к.т.н., доц.*
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від «*28*» *вересня* 2020 року № *4/7-687*

2. Термін подання студентом завершеної роботи *15 грудня 2020*

3. Вихідні дані до роботи *текстові датасети англійською та українською мовами*

4. Зміст роботи (перелік питань, які потрібно розробити) *1. ВСТУП*
2. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ
3. ОБҐРУНТУВАННЯ ОБРАНИХ ЗАСОБІВ
4. РЕАЛІЗАЦІЯ СИСТЕМИ ПЕРЕКЛАДУ ПРИРОДНОЇ МОВИ НА ОСНОВІ МОДЕЛІ "ПОСЛІДОВНІСТЬ-ПОСЛІДОВНІСТЬ" ТА НЕЙРОМЕРЕЖЕВОЇ АРХІТЕКТУРИ ТРАСНФОРМЕРС
5. ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1 - Тема кваліфікаційної роботи магістра
2 - Задачі
3 - Архітектури LSTM і Attention Mechanism
4 - Оцінка вартості реалізації комп'ютерної системи
5 - Повний процес навчання нейромережевої моделі машинного перекладу
6 - Процес навчання
7 - Результат тренування моделі
8 - Висновки

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Охорона праці	*Осухівська Г.М., доц.*		
Безпека в надзвичайних	*Стадник І. Я., проф.*		
ситуаціях			

7. Дата видачі завдання *30 вересня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1	*Аналіз останніх досліджень у галузі обробки природної*	*30.09.2020 -*	*виконано*
	мови та штучних нейронних мереж	*15.10.2020*	
2	*Обґрунтування та розробка нейромережевої*	*16.10.2020 -*	*виконано*
	архітектури для перекладу природної мови	*30.10.2020*	
3	*Формування тестових датасетів та реалізація*	*02.11.2020 -*	*виконано*
	створеної моделі	*13.11.2020*	
4	*Тренування створеної нейромережі та оцінювання*	*16.11.2020 -*	*виконано*
	результатів її роботи. Реалізація системи загалом	*24.11.2020*	
5	*Виконання розділу "Охорона праці та безпека в*	*25.11.2020 -*	*виконано*
	надзвичайних ситуаціях"	*04.12.2020*	
6	*Оформлення пояснювальної записки та графічного*	*07.12.2020 -*	*виконано*
	матеріалу	*11.12.2020*	
7	*Попередній захист кваліфікаційної роботи магістра*	*15.12.2020*	*виконано*
8	*Захист кваліфікаційної роботи магістра*	*23.12.2020*	

Студент _____ *Луцишин Р. О.*
(підпис) (прізвище та ініціали)

Керівник роботи _____ *Луцків А. М.*
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Методи автоматизованого перекладу природної мови на основі нейромережевої моделі “послідовність-послідовність” // Кваліфікаційна робота магістра // Луцишин Роман Олегович // ТНТУ, Комп’ютерна інженерія, група СІм-61 // Тернопіль, 2020 // с. - ___ , рис. - 55, табл. - 1, бібліографія - 26.

Ключові слова: корпус, нейромережева архітектура, глибоке навчання, енкодер, декодер.

Дипломну роботу магістра присвячено дослідженню та реалізації методів автоматизованого перекладу природної мови на основі нейромережевої моделі “послідовність-послідовність”.

Розглянуто основні принципи та підходи до підготовки тренувальної вибірки даних, у тому числі з використанням глибоких нейронних мереж у якості енкодерів. Досліджено та проаналізовано наявні методи вирішення задачі перекладу природної мови, зокрема, було розглянуто декілька нейромережевих архітектур глибокого машинного навчання.

Наведено приклади створення та обробки корпусів природної мови для вирішення задачі формування тренувальної та тестувальної вибірок даних. Було проведено повну оцінку вартості створення комп’ютерної системи, необхідної для вирішення поставленого завдання, а також описано повний процес розгортання програмного забезпечення на даному середовищі за допомогою сторонніх платформ.

Результатами наукового дослідження стали повний огляд існуючих рішень для вирішення поставленої задачі, вибір найкращої технології, вдосконалення останньої, реалізація та тренування глибокої нейромережевої моделі типу “послідовність-послідовність” для задачі перекладу природної мови.

ANNOTATION

Methods of natural language computer-aid translation based on neuro network model of “sequence-sequence” type // Diploma thesis Master degree // Lutsyshyn Roman O. // Ternopil’ Ivan Pul’uj National Technical University, Computer engineering, SIm-61 group // Ternopil, 2020 // P. - ___ , Diagrams - 55, Tables - 1, References - 26.

Keywords: corpus, neural network architecture, deep learning, enoder, decoder.

The master's thesis is devoted to the research and implementation of methods of automated translation of natural language on the basis of the neural network model "sequence-sequence".

The basic principles and approaches to the preparation of training data sampling, including the use of deep neural networks as encoders, are considered. The existing methods of solving the problem of natural language translation have been studied and analyzed, in particular, several neural network architectures of deep machine origin have been considered.

Examples of creation and processing of natural language corpora to solve the problem of forming training and test data samples are given. A full assessment of the cost of creating a computer system required to solve the problem was performed, as well as a complete process of deploying software in this environment using third-party platforms.

The results of the research were a complete review of existing solutions to solve the problem, choosing the best technology, improving the latter, implementation and training of a deep neural network model such as "sequence-sequence" for the problem of natural language translation.

СПИСОК СКОРОЧЕНЬ ТА ТЕРМІНІВ

LSTM (англ. Long Short Term Memory) - тип архітектури глибокого машинного навчання для вирішення задач типу “послідновність-послідовність”, вдосконалена версія RNN;

RNN (англ. Recurent Neural Network) - тип архітектури глибокого машинного навчання для вирішення задач типу “послідновність-послідовність”;

Transformers - тип архітектури глибокого машинного навчання, спрямований вдосконалити LSTM і RNN через використання Attention mechanism (англ. механізм самозвернення);

Корпус - набір текстових даних, представлений у одному файлі;

Токен - одиниця текстової інформації, у даному випадку - слово;

Токенізація - процес створення токенів (у тому числі розбивання на токени).

ЗМІСТ

ВСТУП

Із кожним днем інформаційні технології набувають все більшої і більшої популярності, адже передумовою цьому стало можливе з одного боку спрощення, а з іншого - вдосконалення, виконування повсякденних завдань. Таким чином, ми зараз живемо у епоху пропагування ІТ та альтернативних рішень уже відомим задачам.

Одне із найбільш важливих завдань у процесі технологічного прогресу є взаємодія людини з обчислювальною машиною, саме це стало поштовхом до розвитку такого напрямку у інформатиці як NLP(Natural Language Processing), іншими словами - обробки природної мови. Насправді, термін NLP був введений у вжиток досить давно, проте через недостатньо сильний технологічний рівень ресурсів на той момент часу, люди не змогли повною мірою реалізувати задумане – навчити машину інтелектуально розуміти людський голос та виконувати будь-які операції з розпізнаним текстом.

Актуальністю теми зумовлена необхідністю розробки автоматизованих систем перекладу природної мови. Зокрема, у вирішенні задачі репрезентації текстових даних у числову та векторну форми.

Метою наукового дослідження є розробка методів та засобів автоматизованого перекладу природної мови на основі нейромережевої моделі “послідовність-послідовність” Transformers.

Об’єктом дослідження є нейромережева модель “послідовність-послідовність” Transformers.

Предметом дослідження є нейромережеві моделі та методи автоматизованого перекладу природної мови.

Наукова новизна базується на зміні архітектури нейромережевої моделі Transformers шляхом зміни кількості шарів енкодера та декодера, що зумовило підвищення ефективності процесів навчання та перекладу. Розроблено процедуру формування навчальної та тестувальної вибірок із використанням підходу семантичного аналізу текстів природної мови. Отримано модель глибокого машинного навчання для вирішення задачі перекладу текстів природної мови.

Результати дослідження апробовано на п’яти конференціях:

1. X Всеукраїнська студентська науково-технічна конференція "Природничі та гуманітарні науки. Актуальні питання”.
2. Міжнародна студентська науково-технічна конференція "Природничі та гуманітарні науки. Актуальні питання”.
3. VII Міжнародна науково-технічна конференція молодих учених та студентів "Актуальні задачі сучасних технологій”.
4. XIII Всеукраїнська науково-практична конференція “ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ПРОФЕСІЙНІЙ ДІЯЛЬНОСТІ”
5. International Research-to-Practice Conference for Translators, Young Scholars and Students “Translation Industry: Theory in Action”

Робота складається з пояснювальної записки та графічної частини. Пояснювальна записка складається зі вступу, 4 розділів, висновків, списку використаних джерел та X додатків. Обсяг роботи: пояснювальна записка - 130 аркушів формату А4, графічна частина - 10 аркушів А1.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Постановка задач науково-дослідницької роботи

Незважаючи на постійний прогрес комп’ютерних систем та інформаційних технологій в цілому, обсяги даних, які потрібно опрацьовувати, зберігати та передавати, також збільшуються. Саме тому слід чітко розуміти, що створення підходу, який зміг би зменшити ресурсозатрати та покращити якість раніше запропонований рішень є актуальним завданням нашого часу.

Об’єктом даної роботи є нейромережева модель “послідовність-послідовність” Transformers, предметом - нейромережеві моделі та методи автоматизованого перекладу природної мови.

Мета наукового дослідження - розробити методи та засоби автоматизованого перекладу природної мови на основі нейромережевої моделі “послідовність-послідовність” Transformers.

Серед основних задач даного дослідження слід виокремити наступні:

1. На основі аналізу підходів відображення текстових даних у векторну форму обґрунтувати вибір оптимальних за критерієм розмірності та ефективності паралельного опрацювання.
2. На основі аналізу недоліків існуючих нейромережевих моделей та методів, що використовуються для задач автоматизованого перекладу природної мови сформулювати вимоги до побудови власної нейромережевої моделі.
3. Опрацювати та сформувати навчальні та тестові вибірки текстових даних.
4. Обґрунтувати вибір програмного та апаратного забезпечення реалізації нейромережевої моделі у паралельних та розподілених системах.
5. Реалізувати запропоновану модель у вигляді комп’ютерної системи та здійснити верифікацію запропонованих підходів.

1.2 Аналіз існуючих методів та засобів розв’язання науково-технічних проблем кодування текстових даних

Для вирішення поставлених у дипломній роботі задач було проаналізовано існуючі методи та підходи, які здатні розв’язати науково-технічні проблеми роботи. Оскільки, основним напрямком досліджень є кодування текстової інформації, слід чітко визначити поняття кодування та представлення даних загалом.

Кодування — процес заміни коду даних у текстовому форматі; заміна останніх скороченими та умовними позначеннями; репрезентація інформації будь-якого виду, вираженої засобами природної мови, у послідовність різноманітних символів.

Розрізняють наступні види кодів:

- алфавітні;
- алфавітно-цифровий;
- цифровий.

Станом на сьогодні, найбільш популярними способами кодування інформації є так звані BPE(bytes pair encoding) алгоритми, які передбачають заміну пари символів тексту на один попередньо визначений символ (зазвичай це спеціальні позначення, як-от: @, $, %, & і т.д.).

Таким чином, аналізуючи найпопулярніші підходи до кодування тексту, було розглянуто декілька закордонних статей, які описують використання BPE-based алгоритмів для вирішення різного роду задач, зокрема: задачі нейронного машинного перекладу, розпізнавання власних імен і т.п. Під час проведення аналізу уже існуючих підходів було розглянуто та протестовано деякі з найбільш поширених серед BPE. Справді, використання кодування типу заміни пари байтів на попередньо заданий символ значно зменшує час опрацювання вже закодованих даних [25]. Проте маємо підстави вважати, що діаметрально протилежний підхід до репрезентації опрацьованої інформації, представленої у текстовому вигляді здатний ще більше покращити вже наявні результати. Саме тому було обрано нейронні мережі, як область дослідження більш складних та семантикобазованих способів репрезентації тестових даних у їх числову відповідність.

Архітектура нейронної мережі передбачає використання як мінімум трьох шарів: вхідного, обчислювального та вихідного. Проте між вхідним та обчислювальними шарами існує так званий кодер(механізм представлення вхідних даних для подальшого обчислення в мережі) і декодер(протилежний попередньому механізм). Саме структура кодера-декодера повинна дати більш якісний результат при репрезентації даних із розрахунку подальшої швидкодії опрацювання останніх.

1.3 Проведення теоретичних досліджень різновидів кодування текстових даних

Для дослідження задачі кодування даних було обрано нейронні мережі, основною особливістю яких у даному ключі є архітектурна будова за типом кодер-декодер. Використання архітектурних особливостей нейронних мереж передбачає повне представлення та розуміння останніх.

Нейронні мережі - це обчислювальні системи, які побудовані за принципами побудови біологічних нейронних мереж, що формують мозок тварин. Такі системи навчаються задачам (поступово покращуючи свою продуктивність на них), розглядаючи приклади, загалом без спеціального програмування під конкретне завдання.

Наприклад, у розпізнаванні зображень вони можуть навчатися ідентифікувати зображення, які містять котів, аналізуючи приклади зображень, мічені як «кіт» і «не кіт», і використовувати ці результати для подальшого ідентифікування котів в інших зображеннях. Вони роблять це без жодного апріорного знання про котів. Ключовою відмінністю підходу машинного навчання від загальноприйнятого програмування є повна протилежність суті обидвох парадигм. Таким чином, метою розв’язку задачі у класичному програмуванні є знаходження кінцевого результату при наявності вхідних даних і правил обчислення цього результату. Теорія машинного навчання використовує інший підхід: за наявності вхідних і вихідних даних, слід визначити правила, за якими обчислюються дані виходу.

Такий підхід до розв’язку задачі здатний якнайкраще підібрати правильний спосіб до репрезентації даних, які необхідні для подальших маніпуляцій над ними.

Для проведення досліджень у контексті кодування текстових даних було виконано огляд та аналіз архітектур різнотипних нейронних мереж, які визнаються найбільш досконалими на сьогоднішній день. Зокрема були розглянуті нейромережеві архітектури типу: Neural Network with Attention Mechanism. Transformer-based Neural Network і Neural Network with LSTM layer.

LSTM(long short-term memory) - це архітектура рекурентних нейронних мереж (РНН), що використовується в deep learning (галузі глибокого навчання). На відміну від стандартних нейронних мереж прямого зв'язку, LSTM має з'єднання зворотного зв'язку. У рамках такого підходу можна не тільки обробляти окремі точки даних (наприклад, зображення), але й цілі послідовності даних (наприклад, мовлення або відео). Зокрема, LSTM застосовується до таких завдань, як несегментоване, підключене розпізнавання рукописного вводу, розпізнавання мови та виявлення аномалій у мережевому трафіку (системи виявлення вторгнень) [26].

Мережі LSTM добре підходять для класифікації, обробки та прогнозування на основі даних часових рядів, оскільки між важливими подіями часового ряду можуть бути затримки невідомої тривалості. LSTM мережі були розроблені для вирішення проблеми згасаючого градієнта, з якою можна зіткнутися при навчанні традиційних РНН. Відносна нечутливість до довжини послідовності є перевагою LSTM перед РНН, прихованими моделями Маркова та іншими методами навчання послідовності у чисельних методах. Принцип архітектурної будови однієї LSTM комірки наведено на рисунку 1.1:

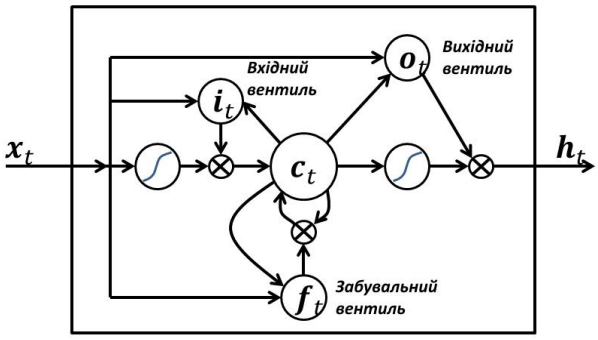

Рис. 1.1 - Будова комірки LSTM

Основним конкурентом LSTM-based нейронної мережі на даний момент є так звані моделі з Attention mechanism, які демонструють кращий результат ніж архітектури попередніх поколінь за рахунок специфікації методів звернення до попередньо надходжених даних.

Attention Mechanism (механізм уваги) - дозволяє декодеру відвідувати різні частини вихідних даних на кожному кроці генерації останніх. Замість того, щоб кодувати послідовність введення в один фіксований контекстний вектор, модель дозволяє дізнатися, як генерувати контекстний вектор для кожного етапу часу виходу. Схематично принцип роботи такої архітектури наведено на рисунку 1.2:

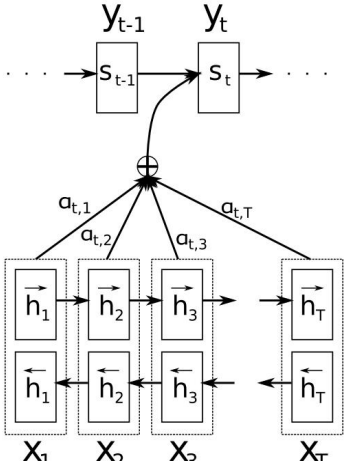

Рис. 1.2 - Принцип роботи Attention Mechanism

Незважаючи на високу якість роботи двох вищенаведених архітектурних рішень для побудови нейронної мережі, станом на сьогодні, кращі результати на великих об’ємах даних показують мережі, побудовані на архітектурі Transformer [2].

Архітектура типу Transformer відрізняється від попередньо згаданої лише тим, що вона містить більш вдосконалений механізм, під назвою Multi-Head Attentions, тобто здатна звертатись до попередніх значень вхідних і вихідних даних багато разів одночасно. Таким чином, здатність “утримувати” контекст значно збільшилась. Будову архітектури Transformer наведено на рисунку 1.3:

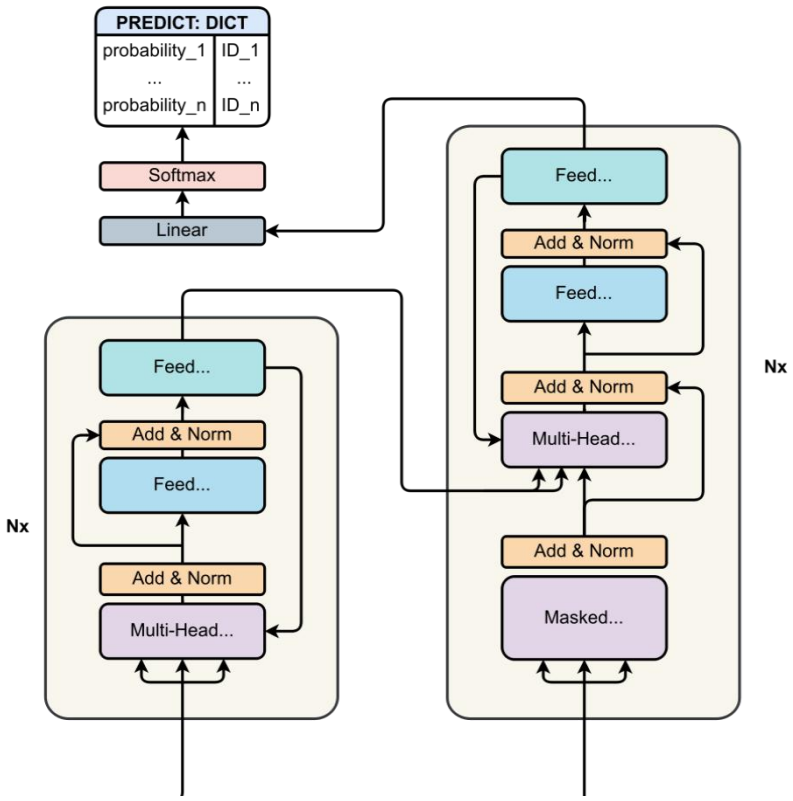

Рис. 1.3 - Вигляд архітектури Transformer

Не можна стверджувати, що якась із вищенаведених архітектур є єдино правильною для використання, або найкращою. Це лише різновиди представлень будови і роботи нейронів у нейронній мережі, які мають своє вузьке застосування під конкретні задачі. Зокрема, різнотипні завдання, як-от: класифікація зображень, генерування тексту чи нейронний машинний переклад, - потребують різного підходу, а як і наслідок - різних архітектурних рішень.

1.4 Огляд засобів для взаємодії із природною мовою

Для того, щоб використовувати будь-яку із вищеперерахованих архітектур чи підходів для вирішення задачі машинного перекладу, потрібно правильно взаємодіяти із даними природної мови, зокрема провести структурний та статистичний аналізи, видалити усе непотрібне з вхідного тексту та інше. Таким

чином, нижче наведено огляд бібліотек та засобів для роботи із текстами природної мови.

1.4.1. Розгляд пакету інструментів для роботи з природною мовою Natural Language Toolkit. NLTK(Natural Language Toolkit) - це платформа, що використовується для побудови програм текстового аналізу. Платформа була спочатку випущена Стівеном Бердом та Едвардом Лопером у поєднанні з курсом обчислювальної лінгвістики в Університеті Пенсільванії в 2001 році. Існує супровідна книга для цього продукту під назвою Natural Language Processing with Python.

Дана бібліотека дає змогу використовувати більшість стандартних рішень для обробки тексту, зокрема його поділу на токени(переважно слова), завантаження готових текстових корпусів обраної мови та багато іншого. Нижче наведено найчастіше вживані варіанти використання NLTK.

Одним із перших кроків до аналізу та опрацювання текстових даних є його поділ на токени, тобто на розділені однакові за структурою фрагменти тексту, ними можуть бути слова, речення, а деколи навіть придумані власні патерни поділу за спеціальними символами. Оскільки, стандартні засоби Python, зокрема split(string, pattern), дають можливість проводити розподіл за власними сталими виразами, то засобами NLTK можна обрати готовий токенайзер, в який на вхід потрібно подати лише потрібну стрічку, наприклад:

- word_tokenize - токенізація за словами, незалежно від розділових знаків;
- wordpunct_tokenize - токенізація за словами, проте з урахуванням знаків пунктуації - крапка, кома, двокрапка і т.п.;
- sent_tokenize - токенізація за реченнями, незалежно від розділових знаків;
- whitespaceTokenizer - токенізація зі створенням кортежу ідентифікаторів токена.

Інколи для завдання аналізу тексту потрібно порахувати частоту вживання того, чи іншого слова, і для того, щоб не придумувати та писати власні алгоритми пошуку слова в тексті, його повторення та підрахування кількості повторень, існує функція

FreqDist(text1), бібліотеки NLTK, яка розташовує найбільш вживані слова у порядку спадання.

При подальшій роботі з текстовими корпусами потрібно привести його до нормальної форми, іншими словами лематизувати, для цього використовують чимало методів, але у бібліотеці NLTK є власні засоби для реалізації лематизування.

Першим із прийомів приведення тексту до нормальної форми є усунення так званих “стоп слів”, тобто усіх займенників, прийменників та інших службових частин мови, у зв’язку з повільним розвитком української нейролінгвістики усі приклади будуть наведені англійською. На рисунку 1.4 зображений приклад усунення “стоп слів” з тексту:

```
# nltk.download('stopwords')
from nltk.corpus import stopwords
stop_words= set(stopwords.words('english'))
# stop_words= set(stopwords.words('russian'))
print ('Total len : {}'.format(len(list(stop_words))))
print(list(stop_words)[:50])
```

```
Total len : 179
['haven', 'before', 'should', 'won', 'this', "that'll", 'can', 'some', 'does', "shan't", 'too',
'about', 's', 'had', 'who', 'from', 'there', "you'd", 'the', 'our', 'that', 'into', 'being', 'h
e', 'needn', 'out', 'same', 'weren', 'ours', "she's", 'which', "doesn't", 'were', 'yourselves',
'an', 'so', 'further', 'where', 'of', 'not', "didn't", 'through', 'theirs', 'no', "mustn't", 'ver
y', 'myself', 'm', 'her', 'each']
```

Рис. 1.4 - Видалення “стоп слів” із тексту

```
from nltk.stem import PorterStemmer
from nltk.tokenize import  word_tokenize
import numpy as np

target_text = 'This is an example showing the stop words filtering. Hello Mr. Smith! How are you
 doing today? The whether is great and the Python is awesome. The sky is blue '

words= np.array(word_tokenize(target_text))
ps= PorterStemmer()
v_stem= np.vectorize(ps.stem)
stemmed_words= v_stem(words)

print (target_text)
stemmed_words
```

```
This is an example showing the stop words filtering. Hello Mr. Smith! How are you doing today? Th
e whether is great and the Python is awesome. The sky is blue
array(['thi', 'is', 'an', 'exampl', 'show', 'the', 'stop', 'word',
       'filter', '.', 'hello', 'mr.', 'smith', '!', 'how', 'are', 'you',
       'do', 'today', '?', 'the', 'whether', 'is', 'great', 'and', 'the',
       'python', 'is', 'awesom', '.', 'the', 'sky', 'is', 'blue'],
      dtype='<U7')
```

Рис. 1.5 - Усунення усіх значущих частин слова, окрім кореня

Як видно з рисунка вище, видалені слова не несуть великої інформативності для подальшого кількісного чи якісного аналізу тексту.

Наступним етапом у нормалізації тексту є усунення усіх закінчень, префіксів і суфіксів для залишення лише кореня слова, приклад наведено вище, на рисунку 1.5. Досить важливою частиною аналізу тексту є визначення частини мови того чи іншого слова, для цього завдання NLTK має також готове рішення. Після використання цього рішення обраний текст чи стрічка буде мати наступний вигляд: слово – тег(позначення частини мови), інколи визначення POS(part of speech) є важливим для виокремлення тих чи інших слів, відповідно до потреб.

Задля покращення пошуку та взаємодії з текстом часто використовують визначення власних назв – NER(named entity recognition), NLTK має засоби для вирішення даного завдання, проте вони значно гірші, аніж у інструмента flair, опис і приклад використання якого буде наведено у останньому розділі.

Вирішення поставлених завдань у сфері NLP часто потребує пошуку синонімів, саме для цих потреб дана бібліотека має можливість виконувати пошук подібних слів у вже натренованій моделі синонімів WordNet, приклад використання наведено у рисунку 1.6:

```
target_word = 'look'
print('target_word = {} '.format(target_word))

synsets = wn.synsets(target_word)
print (synsets[0])
[synset.name() for  synset  in synsets]
```

```
target_word = look
Synset('expression.n.01')
['expression.n.01',
 'look.n.02',
 'look.n.03',
 'spirit.n.02',
 'look.v.01',
 'look.v.02',
 'look.v.03',
 'search.v.02',
 'front.v.01',
 'attend.v.02',
 'look.v.07',
 'expect.v.03',
 'look.v.09',
 'count.v.08']
```

Рис. 1.6 – Пошук синонімів до заданого слова за допомогою моделі WordNet

На наведеному вище рисунку видно, що синоніми зображені також з частиною мови та порядковим номером у цій мережі для обраного слова. Окрім пошуку синонімів мережа WordNet дає змогу отримати коротке обґрунтування значення того, чи іншого слова, що також може бути використаним для багатьох завдань [20].

Edit Distance(відстань редагування) – число, яке вказує на те, скільки перестановок літер потрібно зробити, щоб слово А стало словом Б. Часто даний прийом використовується для задач spellchecking, тобто перевірки правильності вводу користувачем стрічки. Приклад використання відстані редагування наведено нижче, на рисунку 1.7:

```
target_pairs= [
    ('hello', 'hell'),
    ('hell', 'hall'),
    ('men', 'manual'),
    ('муха', 'слон'),
    ('casual', 'causal'),
    ('top', 'pot'),
    ('top', 'open')

]
for word_1, word_2 in target_pairs:
    print ('{}, {}: {}/{}'.format(word_1, word_2, nltk.edit_distance(word_1, word_2,transposition
s=True),
                                   nltk.edit_distance(word_1, word_2,transpositions=False)))
```

```
hello, hell: 1/1
hell, hall: 1/1
men, manual: 4/4
муха, слон: 4/4
casual, causal: 1/2
top, pot: 2/2
top, open: 3/3
```

Рис. 1.7 – Відстань редагування між двома словами

Відповідно до вищезгаданої відстані редагування також існує поняття семантичної подібності двох слів, таким чином уже нам відома мережа WordNet дає змогу в числовому вигляді дати уявлення про смислову подібність двох слів, насправді визначення цієї подібності вираховується за відповідністю відстані обидвох понять у ієрархічному дереві та кількості перестановок, потрібних для перетворення одного слова в інше. Забігаючи наперед, слід сказати, що семантики в даному методі не надто багато, тому для вирішення задачі семантичного змісту слова використовуються техніки Word Emmbeddings, зокрема Word2Vec(для слів) та Doc2Vec(для документів), але про це згодом. Наприклад, завдяки мережі WordNet можна отримати, що “Олень” подібний до “Коня” з ймовірністю 80%, що звичайно

відповідає розташуванню цих тварин у ієрархії світової фауни, проте не має значного сенсу, адже це різні тварини [6].

Бібліотека NLTK дає змогу вирішувати набагато більше задач, але найбільш популярні та затребувані – наведені вище.

1.4.2 Word Embeddings – представлення слів у формі числового вектора. Якщо мова йде про пошук семантики, або використання нейронних мереж для задач вирівнювання паралельних корпусів або нейронного машинного перекладу, одразу ж виникає проблема представлення текстового формату даних. Для вирішення цього існує декілька способів, нижче буде наведено основні з них.

Векторна модель – це алгебраїчна модель для представлення текстових даних (і будь-яких об'єктів загалом) у вигляді векторів ідентифікаторів (таких як індекси терміну і т.п.). Вона використовується для фільтрації та пошуку інформації, індексації та створення рейтингу відповідності.

Документ у векторній моделі розглядається як невпорядкована множина термів. Термами в інформаційному пошуку називають слова, з яких складається текст, а також такі елементи тексту, як, наприклад, 2010, II-5 або Тянь-Шань.

Різними способами можна визначити вагу терма в документі — «важливість» слова для ідентифікації цього тексту. Наприклад, можна просто підрахувати кількість вживань терма в документі, так звану частоту терма, — чим частіше слово зустрічається в документі, тим більша у нього буде вага. Якщо терм не зустрічається в документі, то його вага в цьому документі дорівнює нулю.

Всі терми, які трапляються в документах оброблюваної колекції, можна впорядкувати. Якщо тепер для деякого документа виписати по порядку ваги всіх термів, включаючи ті, яких немає в цьому документі, вийде вектор, який і буде представленням цього документа у векторному просторі. Розмірність цього вектора, як і розмірність простору, дорівнює кількості різних термів у всій колекції, і є однаковою для всіх документів. Більш наглядно це можна зобразити формулою, зображеною на рисунку 1.8:

$$d_j = (w_{1,j}, w_{2,j}, \ldots, w_{t,j})$$
$$q = (w_{1,q}, w_{2,q}, \ldots, w_{n,q})$$

Рис. 1.8 - Формальне представлення ваги терма

Застосування векторної моделі може бути дуже різним, проте найчастіше - це пошук семантичної схожості двох векторів, яка обчислюється за допомогою звичайної косинус-відстані(значення косинуса між двома векторами), таким чином результатом стає ймовірність, із якою обидва вектори, а як і наслідок, слова подібні між собою.

Існує чимало варіантів векторного представлення слів та цілих документів, нижче наведено перелік найпопулярніших із них.

TF (term frequency - частота слова) - це відображення відношення числа входження обраного терміна (слова) до загальної кількості термінів (слів) у документі. Таким чином, оцінюється важливість терміна (слова) t_i у межах обраного документа. Нижче наведена формула для обчислення частоти слів:

$$\mathrm{TF} = \frac{\mathrm{n_i}}{\sum_{\mathrm{k}} \mathrm{n_k}} \tag{1.1}$$

У вищенаведеній формулі n_i є числом входжень слова в документ, а в знаменнику знаходиться загальна кількість слів в документі.

IDF (inverse document frequency - обернена частота документа) - протилежність частоти, із якою термін (слово) зустрічається у тексті. Використання IDF зменшує кількість частовживаних слів. Розрахунок протилежної частоти документа проводиться за наступною формулою:

$$\mathrm{IDF} = \log \frac{|D|}{|d_i \supset t_i|} \tag{1.2}$$

У знаменнику цієї формули знаходиться кількість документів, в яких зустрічається слово t_i (коли $n_i \neq 0$).

Наступна модель векторного представлення називається BOW(bag of words), або мішок із словами. Модель bag-of-words є спрощеним поданням, що використовується в NLP і пошуку інформації (IR - information research). У цій моделі текст (наприклад, речення або документ) представляється у вигляді мішка (мультимножини) його слів, зневажаючи граматику і навіть порядок слів, але зберігаючи кратність. Модель «мішок слів» також використовується для задач розпізнавання графічних об’єктів.

Дана модель зазвичай використовується в методах класифікації документів, де частота виникнення кожного слова використовується як ознака для навчання класифікатора.

Принцип роботи мішка слів простий і зрозумілий, насамперед визначаються усі унікальні слова у тексті, після чого будується з них словник. Відповідно до кожної стрічки кожне слово з цього словника буде мати два можливих варіанти представлення: 0 - якщо слово зі словника не знаходиться у даній стрічці та 1 - якщо знаходиться. Таким чином ми отримуємо набір двійкових векторів, які репрезентують кожну стрічку з текстового корпусу у числовому вигляді. Приклади використання векторних моделей, базованих на використанні BOW будуть наведені у подальших розділах.

Останнім, найбільш відомим представленням тексту у векторній формі є skipgramms. Принцип роботи даного методу полягає у формуванні групи токенів сталої довжини, в яких відповідне слово поєднується кортежем із іншим словом з групи, таким чином ми отримаємо набір кортежів, що слугує потім для перетворення цього набору в вектор. Детальніше ознайомитись із алгоритмом роботи технології skipgramm можна на рисунку 1.11:

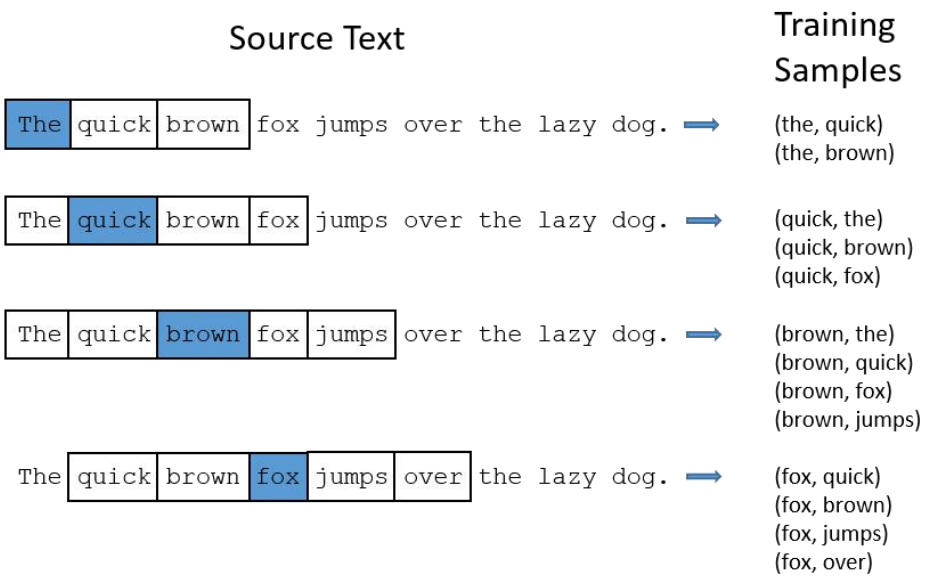

Рис. 1.11 - Алгоритм роботи технології skipgramm

Отож, підсумовуючи все вищесказане, слід зауважити, що якість пошуку семантично подібних слів залежить не стільки від формули обчислення кута між векторами, як від моделі векторного представлення цього слова. Детальніше порівняння та оцінка якості векторного представлення тексту будуть наведені в наступному розділі.

1.4.3 Пошук власних назв у стрічці(named entity recognition). Для визначення частини мови потрібного слова уже було запропоновано використовувати засоби з бібліотеки NLTK, проте варто зазначити, що часто виникає потреба виокремлення власних назв, зокрема їхнього типу. Із цим завданням вищезгадана бібліотека справляється не дуже добре, оскільки для цього більш правильним було б використання спеціально тренованої моделі глибокого машинного навчання. Такою моделлю є Flair.

Flair - модель глибокого машинного навчання, призначена для визначення власних назв та їх типу, наприклад: локація, персона, компанія та інші. Flair - була розроблена інженерами компанії Facebook для їх внутрішніх потреб, проте також знайшла своє застосування у широкому загалі, де задача типізації власних назв вимагала точності для більшості мов світу [9]. Процес визначення NER(named entity

recognition) достатньо складний, проте варто відмітити основні моменти зарахування слова до власної назви. Отже, одним із ключових етапів є перевірка першої літери на регістр, після чого виконується врахування позиції кожного слова. Ці дані передаються у нейронну мережу, яка потім дає змогу більш точно вказувати на тип власної назви слова. Приклад тегування речення наведений нижче, на рисунку 1.12:

```
sentence = Sentence('George Washington went to Washington .')

# predict NER tags
tagger.predict(sentence)

# print sentence with predicted tags
print(sentence.to_tagged_string())
```

This should print:

```
George <B-PER> Washington <E-PER> went to Washington <S-LOC> .
```

Рис. 1.12 - Приклад використання утиліти flair

Насправді, застосування тегування власних назв дуже широке та не обмежується звичайним виводом тега відповідно до конкретного поняття, у перспективі можливе використання NER задля покращення якості пошуку семантичних дублікатів між двома стрічками.

Останнім, та, мабуть, найбільш серйозним завданням у сфері NLP є машинний переклад, адже саме він потребує чимало ресурсів. Машинний переклад - технології автоматизованого перекладу текстів (письмових та усних) з однієї природної мови на іншу за допомогою комп'ютера; напрямок наукових досліджень, пов'язаний з побудовою систем автоматизованого перекладу.

На базовому рівні, робота комп'ютерних програм для перекладу полягає у заміні слів чи словосполучень з однієї мови на слова чи словосполучення з іншої. Однак тоді виникає проблема, що така заміна не може забезпечити якісний переклад тексту, адже потрібне визначення та розпізнання слів та цілих фраз з мови оригіналу. Це спонукає активну наукову діяльність у галузі комп'ютерної лінгвістики. Наразі,

для вирішення неоднозначностей при перекладі, використовуються багатомовні онтологічні ресурси, такі як WordNet та UWN, а також нейронний машинний переклад, який базується на основі багатошарових глибоких нейронних мереж, переважно це рекурентні нейронній мережі.

Поглиблене навчання або поглиблене засвоєння інформації (Deep learning) (також відоме як глибоке структурне навчання або ієрархічне навчання) є частиною більш широкої групи методів машинного навчання, що базуються на інтерпретації результатів навчання, на відміну від алгоритмів конкретних завдань. Навчання може бути контрольованим, або без нагляду. Огляд реального прикладу використання моделей глибоких нейронних мереж для нейронного машинного перекладу буде наведено у останньому розділі.

1.4.4 Використання Word2Vec і Doc2Vec моделей. У цьому розділі будуть розглянуті основні види реалізацій моделей векторного представлення тексту, зокрема, це - Word2Vec, Doc2Vec, USE. Відмінність між моделями полягає у використанні різних підходів до перетворення тексту у вектор і у реалізації різних архітектурних рішень для них.

Word2vec - набір моделей для аналізу семантики природних мов, що представляє собою технологію, яка заснована на дистрибутивній семантиці і векторному поданні слів. Цей інструмент був розроблений групою дослідників Google в 2013 році [19].

Робота цієї технології здійснюється наступним чином: word2vec приймає великий текстовий корпус в якості вхідних даних і зіставляє кожному слову вектор, видаючи координати слів на виході. Спочатку він створює словник, «навчаючись» на вхідних текстових даних, а потім обчислює векторне подання слів. Векторне подання ґрунтується на контекстній близькості: слова, що зустрічаються в тексті поруч з однаковими словами (а отже, мають схожий зміст), у векторному поданні матимуть близькі координати векторів-слів. Отримані вектори-слова можуть бути використані для обробки природної мови та машинного навчання.

Для word2vec розроблені два основних алгоритми навчання: CBoW (Continuous Bag of Words, «безперервний мішок зі словами") і Skip-gram. CBoW - архітектура, яка передбачає поточне слово, виходячи з навколишнього його контексту. Архітектура типу Skip-gram діє інакше: вона використовує поточне слово, щоб передбачати навколишні його слова. Користувач word2vec має можливість перемикатися і вибирати між алгоритмами. Порядок слів контексту не впливає на результат ні в одному з цих алгоритмів. Для роботи з моделлю word2vec потрібно спочатку підготувати дані, тобто завантажити стрічки та токенізувати їх, щоб отримати список токенів(у даному випадку токен - це слово).

Нижче буде приведений приклад застосування моделі word2vec, див. Рис. 1.13:

```
def read_input(input_file):
    """This method reads the input file which is in gzip format"""

    logging.info("reading file {0}...this may take a while".format(input_file))
    with gzip.open(input_file, 'rb') as f:
        for i, line in enumerate(f):

            if (i % 10000 == 0):
                logging.info("read {0} reviews".format(i))
            # do some pre-processing and return list of words for each review
            # text
            yield gensim.utils.simple_preprocess(line)
```

Рис. 1.13 - Завантаження та токенізація текстового корпусу

Після виконання наведених вище дій слід виконати тренування моделі, виконується це просто, завдяки лише одній команді:

```
model = gensim.models.Word2Vec(
    documents,
    size=150,
    window=10,
    min_count=2,
    workers=10,
    iter=10)
```

Рис. 1.14 - Код тренування моделі Word2Vec

Де documents - попередньо приготований текст, size - розмірність вектора, window - максимальна відстань між поточним та передбачуваним словами, min_count - ігнорування усіх слів із загальною частотою вживання менше заданого значення, workers - використання заданої кількості потоків, iter - кількість виконуваних ітерацій(інколи зустрічається epoch).

Після тренування моделі, можна її зберегти, або використати вже попередньо натреновану. Коли модель успішно натренована можна використовувати стандартні методи визначення найбільш схожих слів, використання яких наведено на рисунку 1.15:

```
w1 = "dirty"
model.wv.most_similar (positive=w1)
```

```
2018-01-26 23:03:42,416 : INFO : precomputi
```

```
[('filthy', 0.871721625328064),
 ('stained', 0.7922376990318298),
 ('unclean', 0.7915753126144409),
 ('dusty', 0.7772612571716309),
 ('smelly', 0.7618112564086914),
 ('grubby', 0.7483716011047363),
 ('dingy', 0.7330487966537476),
 ('gross', 0.7239381074905396),
 ('grimy', 0.7228356599807739),
 ('disgusting', 0.7213647365570068)]
```

Рис. 1.15 - Найбільш схожі семантично слова до слова dirty

Як видно на зображенні вище, окрім найбільш схожих слів також фігурують і ймовірності, які слугують кількісною характеристикою подібності між словами. Використання функції передбачення найбільш подібних слів має чимало варіацій, зокрема використання різних вхідних параметрів функції може фільтрувати вихідний результат за позитивністю чи негативністю, а також виводити задану кількість семантично подібних слів:

```
# look up top 6 words similar to 'france'
w1 = ["france"]
model.wv.most_similar (positive=w1,topn=6)
```

```
[('canada', 0.6603403091430664),
 ('germany', 0.6510637998580933),
 ('spain', 0.6431018114089966),
 ('barcelona', 0.61174076795578),
 ('mexico', 0.6070996522903442),
 ('rome', 0.6065913438796997)]
```

Рис. 1.16 - Виведення на екран лише позитивних варіантів подібних слів

Підсумовуючи все вищезгадане, слід зауважити, що використання векторних моделей представлення, зокрема CBOW і SkipGramms дозволяє доволі точно знаходити семантично подібні слова до заданого, проте під час використання цілих речень ситуація сильно змінюється, адже врахування значення окремих слів не може відобразити різницю між стрічками, в яких відмінність полягає в одному слові, так як більшість семантично схожих слів переважає над вагою одного слова.

Для вирішення даної проблеми існує декілька шляхів, зокрема визначення середнього значення векторів для речення та використання більш досконалих архітектурних рішень для цілих речень.

Однією з покращених моделей Word2Vec є Doc2Vec, як зрозуміло з назви, суть роботи останньої полягає в перетворенні в векторну форму представлення не окремих слів, а цілих речень, чи документів(відповідно до заданої довжини вхідного буфера).

Метод Doc2Vec містить у собі два методи: distributed memory (DM, розподілена пам'ять) і distributed bag of words (DBOW, розподілений мішок слів). Метод DM прогнозує слово за відомими попередніми словами і вектором абзацу. Незважаючи на те, що контекст переміщається по тексту, вектор абзацу не рухається (звідси назва «розподілена пам'ять») і дозволяє врахувати порядок слів. DBOW прогнозує випадкові групи слів в абзаці тільки на підставі вектора абзацу.

Використання Doc2Vec дуже схоже до використання попередньої моделі, тому наведено приклад без пояснення аргументів, переданих у вхід функції:

```
>>> from gensim.test.utils import common_texts
>>> from gensim.models.doc2vec import Doc2Vec, TaggedDocument
>>>
>>> documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(common_texts)]
>>> model = Doc2Vec(documents, vector_size=5, window=2, min_count=1, workers=4)
```

Рис. 1.17 - Тренування моделі Doc2Vec

Аналогічно до попереднього випадку, є можливість зберігати та використовувати вже попередньо натреновану модель, адже інколи ресурси не дозволяють проводити тренування на достатній кількості даних, тому існує чимало вже натренованих моделей Word2Vec та Doc2Vec. Лідером у цьому напрямі, звісно, є компанії Google та Facebook, адже обидві мають змогу тренувати свої програмні рішення на відповідному апаратному забезпеченні, а також на відповідних текстових ресурсах різних мов світу.

1.4.5 Огляд моделі глибокого машинного навчання BERT. BERT - це метод попередньої підготовки мови, що передбачає готування загальної моделі "розуміння мови" на великому текстовому корпусі (наприклад, на Вікіпедії), а потім використовуємо цю модель для задач NLP. BERT перевершує попередні методи, оскільки це перша модель глибокого машинного навчання, яка базується на unsupervised learning(навчання без вчителя) [3].

Без вчителя означає, що BERT навчався, використовуючи лише звичайний текст, що є важливим, оскільки величезна кількість звичайних текстових даних є загальнодоступною в Інтернеті багатьма мовами, що лише сприяє збору більшої кількості даних.

Попередньо натреновані моделі також можуть бути контекстно-вільними або контекстуальними, а їх контекстні подання можуть бути однонаправленими або двонаправленими. Контекстні моделі, такі як word2vec або GloVe, створюють єдине вектор для кожного слова у словнику, тому вектор зможе дати представлення лише про конкретне одне слово, без урахування інших. Контекстні моделі замість цього створюють представлення кожного слова, яке базується на інших словах у реченні.

Уже натреновані моделі BERT дають змогу виконувати хороший пошук семантичних дублікатів стрічок, адже ресурси компанії Google стали передумовою тренування на дуже великому об’ємі даних, що звичайний користувач собі дозволити не в змозі. Приклад використання моделі BERT для задачі бінарної класифікації відгуків наведено нижче:

```
from tensorflow import keras
import os
import re

# Load all files from a directory in a DataFrame.
def load_directory_data(directory):
  data = {}
  data["sentence"] = []
  data["sentiment"] = []
  for file_path in os.listdir(directory):
    with tf.gfile.GFile(os.path.join(directory, file_path), "r") as f:
      data["sentence"].append(f.read())
      data["sentiment"].append(re.match("\d+_(\d+)\.txt", file_path).group(1))
  return pd.DataFrame.from_dict(data)

# Merge positive and negative examples, add a polarity column and shuffle.
def load_dataset(directory):
  pos_df = load_directory_data(os.path.join(directory, "pos"))
  neg_df = load_directory_data(os.path.join(directory, "neg"))
  pos_df["polarity"] = 1
  neg_df["polarity"] = 0
  return pd.concat([pos_df, neg_df]).sample(frac=1).reset_index(drop=True)

# Download and process the dataset files.
def download_and_load_datasets(force_download=False):
  dataset = tf.keras.utils.get_file(
      fname="aclImdb.tar.gz",
      origin="http://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz",
      extract=True)

  train_df = load_dataset(os.path.join(os.path.dirname(dataset),
                                       "aclImdb", "train"))
  test_df = load_dataset(os.path.join(os.path.dirname(dataset),
                                      "aclImdb", "test"))

  return train_df, test_df
```

Рис. 1.18 - Ініціалізація завантажувача файлів для моделі

Суть задачі класифікації відгуків полягає у визначенні позитивності або ж негативності вказаного відгука на книгу, фільм і т. д., таким чином результатом роботи алгоритму має стати словник значень “відгук - його значення”.

Після завантаження даних, слід їх опрацювати, тобто розподілити на тренувальну та тестувальну множину, відповідно до назв - на тренувальній модель буде тренуватись, а на тестувальній - перевіряти вірність свого передбачення.

```
# Use the InputExample class from BERT's run_classifier code to create examples from the data
train_InputExamples = train.apply(lambda x: bert.run_classifier.InputExample(guid=None, # Globall
y unique ID for bookkeeping, unused in this example
                                                                   text_a = x[DATA_COLUMN],
                                                                   text_b = None,
                                                                   label = x[LABEL_COLUMN]), axis
= 1)

test_InputExamples = test.apply(lambda x: bert.run_classifier.InputExample(guid=None,
                                                                   text_a = x[DATA_COLUMN],
                                                                   text_b = None,
                                                                   label = x[LABEL_COLUMN]), axis
= 1)
```

Рис. 1.19 - Розбиття даних на тренувальну та тестувальну множини

Коли дані підготовані та розбиті на множини, потрібно створювати завантажувач попередньо натренованої моделі:

```
# This is a path to an uncased (all lowercase) version of BERT
BERT_MODEL_HUB = "https://tfhub.dev/google/bert_uncased_L-12_H-768_A-12/1"

def create_tokenizer_from_hub_module():
  """Get the vocab file and casing info from the Hub module."""
  with tf.Graph().as_default():
    bert_module = hub.Module(BERT_MODEL_HUB)
    tokenization_info = bert_module(signature="tokenization_info", as_dict=True)
    with tf.Session() as sess:
      vocab_file, do_lower_case = sess.run([tokenization_info["vocab_file"],
                                            tokenization_info["do_lower_case"]])

  return bert.tokenization.FullTokenizer(
      vocab_file=vocab_file, do_lower_case=do_lower_case)

tokenizer = create_tokenizer_from_hub_module()
```

Рис. 1.20 - Завантаження попередньо натренованої моделі

Насправді, ще чимало кроків і етапів до робочого стану класифікатора відгуків, адже потрібно створити алгоритм обчислення функції втрат, варто визначитись із функцією оптимізації та активації(невід’ємні параметри усіх нейронних мереж), а вже після цього слід визначати тип відгука, результатом роботи завантаженої вище моделі є класифікатор текстових відгуків, зображений на рисунку 1.21:

```
predictions
```

```
[('That movie was absolutely awful',
  array([-4.9142293e-03, -5.3180690e+00], dtype=float32),
  'Negative'),
 ('The acting was a bit lacking',
  array([-0.03325794, -3.4200459 ], dtype=float32),
  'Negative'),
 ('The film was creative and surprising',
  array([-5.3589125e+00, -4.7171740e-03], dtype=float32),
  'Positive'),
 ('Absolutely fantastic!',
  array([-5.0434084 , -0.00647258], dtype=float32),
  'Positive')]
```

Рис. 1.21 - Класифікатор текстових відгуків

1.4.6 Огляд і застосування моделі векторного представлення тексту USE. USE(universal sentence encoder) - модель для кодування речень у векторне представлення, які в подальшому спеціально спрямовані на використання у різних задачах сфери NLP. Модель є ефективною та забезпечує точне виконання різноманітних завдань. Два варіанти моделі кодування дозволяють робити компроміс між точністю і обчислювальними ресурсами. Як показав досвід, для представлення речень, а не слів, у векторній формі - дана модель є найкращою, оскільки її швидкодія та якість перетворення сильно відрізняється від аналогів у кращу сторону. Нижче буде наведено приклад можливого використання USE моделі для перетворення речень у вектори [18].

Для початку потрібно підготувати вхідні дані, нехай це будуть штучно підібрані речення:

```
X = [
    # Smartphones
    "I like my phone",
    "My phone is not good.",
    "Your cellphone looks great.",

    # Weather
    "Will it snow tomorrow?",
    "Recently a lot of hurricanes have hit the US",
    "Global warming is real",

    # Food and health
    "An apple a day, keeps the doctors away",
    "Eating strawberries is healthy",
    "Is paleo better than keto?",

    # Asking about age
    "How old are you?",
    "what is your age?",
]

y = [0,0,0,1,1,1,2,2,2,3,3]
```

Рис. 1.22 - Підготовка даних для моделі USE

Наступним кроком є завантаження попередньо натренованої моделі, для подальшого представлення речень у вектори:

```
from tensorflow.python.saved_model import tag_constants

scope = 'finetune'

graph=tf.Graph()

with tf.Session(graph=graph) as sess:
    model_path = 'model/'
    tf.saved_model.loader.load(sess, [tag_constants.SERVING], model_path)

    sess.run(tf.global_variables_initializer())
    sess.run(tf.get_default_graph().get_operation_by_name('finetune/init_all_tables'))

    in_tensor = tf.get_default_graph().get_tensor_by_name(scope + '/module/fed_input_values:0')
    ou_tensor = tf.get_default_graph().get_tensor_by_name(scope + '/module/Encoder_en/hidden_laye
rs/l2_normalize:0')

    run_and_plot(sess, in_tensor, X, ou_tensor)
```

Рис. 1.22 - Завантаження попередньо натренованої моделі

Далі вже перетворені в вектори речення можна використовувати для різних завдань, зокрема пошуку семантично схожих стрічок(семантичних дублікатів), у наступному розділі буде наведено приклад використання моделі USE та утиліти flair для пошуку семантично подібних стрічок у межах одного текстового корпусу.

Підсумовуючи все вищезгадане, потрібно виокремити ключові моменти, отож: для роботи зі смисловим навантаженням тексту(його семантикою), слід обов’язково перевести його в векторну модель представлення. Важливим також є й те, що від розмірності вектора залежить якість подальшої роботи з ним, адже чим більша розмірність, тим більш інформативним є сам вектор для наступного опрацювання. Саме тому відповідно до поставленого завдання потрібно правильно підібрати спосіб перетворення тексту у вектор, або ж релевантну модель, мета якої полягатиме в оптимізації цього ж процесу.

Після виконання векторного перетворення тексту, у дію вступають формули з теорії математичної статистики та векторної алгебри, які здатні аналітично та навіть візуально відтворити текст у числовому представленні.

1.4.7 Огляд бібліотеки для роботи з текстом GenSim. Аналогічно до згаданої у першому розділі бібліотеки NLTK, яка створена для опрацювання природної мови, також існує хороший аналог під назвою GenSim.

GenSim - готова бібліотека з відкритим вихідним кодом для безлімітного моделювання тематики та обробки природних мов, використовуючи сучасне статистичне машинне навчання.

GenSim реалізована у Python та Cython для максимальної продуктивності та масштабованості. Бібліотека спеціально розроблена для обробки великих колекцій тексту за допомогою потокового передавання даних та інкрементних алгоритмів реального часу, що відрізняє його від більшості інших пакетів програмного забезпечення для машинного навчання, які орієнтовані лише на обробку пам'яті, зокрема від NLTK. Саме GenSim містить у собі вже натреновані моделі Word2Vec, Doc2Vec та інші. Дана бібліотека дає змогу токенізувати текст за різними критеріями, причому цей процес відбувається значно якісніше та швидше ніж у аналогів. Нижче будуть наведені приклади роботи з GenSim.

Отож, одним із перших етапів роботи з текстовими даними в сфері NLP, як зазначалось вище, є токенізація, для цього GenSim пропонує чимало ефективних рішень, зокрема функція gensim.utils.simple_preprocess() виконує безпосередню підготовку текстових даних для подальшого їх передавання у модель Word2Vec чи інші, на рисунку 1.23 зображений процес перетворення стрічки у токени:

```
def read_input(input_file):
    """This method reads the input file which is in gzip format"""

    logging.info("reading file {0}...this may take a while".format(input_f
    with gzip.open(input_file, 'rb') as f:
        for i, line in enumerate(f):

            if (i % 10000 == 0):
                logging.info("read {0} reviews".format(i))
            # do some pre-processing and return list of words for each rev
            # text
            yield gensim.utils.simple_preprocess(line)
```

Рис. 1.23 - Процес токенізації стрічки за допомогою GenSim

Відповідно модель Word2Vec також використаємо зі стандартного пакету GenSim, тому попередній код тренування матиме наступний вигляд:

```
# build vocabulary and train model
    model = gensim.models.Word2Vec(
        documents,
        size=150,
        window=10,
        min_count=2,
        workers=10)
    model.train(documents, total_examples=len(documents), epochs=10)
```

Рис. 1.24 - Код тренування моделі Word2Vec

Очевидно, що на даний момент часу існує чимало засобів та інструментів для роботи з задачами в сфері NLP, що безумовно дає змогу все інтенсивніше розвиватись і набувати популярності в цілому світі.

РОЗДІЛ 2
ОБҐРУНТУВАННЯ ОБРАНИХ ЗАСОБІВ

Для обґрунтування вибору тих чи інших методів і засобів реалізації поставленого завдання було обрано спочатку навести уже існуючі підходи та виявлені в них недоліки, після чого їх вдосконалити. Таким чином, нижче розглянуті найновіші доступні підходи, які застосовуються для вирішення задач цього типу. Зокрема, будуть розглянуті наступні методи / архітектури: Attention Mechanism, Transformers-based моделі.

2.1 Розгляд та побудова нейронної мережі з використанням Attention Mechanism

Незважаючи на вискоку якість та точність усіх архітектур, згаданих у попередньому розділі, слід чітко розуміти, що вони також відрізняються між собою за складністю алгоритму виконання, а тому і навантаження на обчислювальну машину, яка буде запускати ці моделі, буде різним.

Для дослідження роботи нейронної мережі було використано мову програмування Python та фреймворк для неї під назвою Torch [8].

Першим етапом у побудові такої нейронної мережі є імпортування усіх необхідних бібліотек та створення енкодера:

```python
import torch
import torch.nn as nn
from torch import optim
import torch.nn.functional as F
from torch.autograd import Variable

class Encoder(nn.Module):
  def __init__(self, input_size, hidden_size, bidirectional = True):
    super(Encoder, self).__init__()
    self.hidden_size = hidden_size
    self.input_size = input_size
    self.bidirectional = bidirectional

    self.lstm = nn.LSTM(input_size, hidden_size, bidirectional = bidirectional)

  def forward(self, inputs, hidden):

    output, hidden = self.lstm(inputs.view(1, 1, self.input_size), hidden)
    return output, hidden

  def init_hidden(self):
    return (torch.zeros(1+int(self.bidirectional), 1, self.hidden_size),
      torch.zeros(1 + int(self.bidirectional), 1, self.hidden_size))
```

Рис. 2.1 - Імпортування необхідних бібліотек і створення енкодера

Останнім кроком у побудові нейронної мережі типу Attention Mechanism є створення саме механізму звернення уваги:

```python
class AttentionDecoder(nn.Module):
  def __init__(self, hidden_size, output_size, vocab_size):
    super(AttentionDecoder, self).__init__()
    self.hidden_size = hidden_size
    self.output_size = output_size

    self.attn = nn.Linear(hidden_size + output_size, 1)
    self.lstm = nn.LSTM(hidden_size + vocab_size, output_size)
    self.final = nn.Linear(output_size, vocab_size)

  def init_hidden(self):
    return (torch.zeros(1, 1, self.output_size),
      torch.zeros(1, 1, self.output_size))

  def forward(self, decoder_hidden, encoder_outputs, input):
    weights = []
    for i in range(len(encoder_outputs)):
      print(decoder_hidden[0][0].shape)
      print(encoder_outputs[0].shape)
      weights.append(self.attn(torch.cat((decoder_hidden[0][0],
                                          encoder_outputs[i]), dim = 1)))
    normalized_weights = F.softmax(torch.cat(weights, 1), 1)
    attn_applied = torch.bmm(normalized_weights.unsqueeze(1),
                             encoder_outputs.view(1, -1, self.hidden_size))

    input_lstm = torch.cat((attn_applied[0], input[0]), dim = 1)

    output, hidden = self.lstm(input_lstm.unsqueeze(0), decoder_hidden)
    output = self.final(output[0])
    return output, hidden, normalized_weights
```

Рис. 2.2 - Імплементування механізму звернення уваги (Self-Attention)

Таким чином, використання вищенаведеного коду дасть змогу не лише кодувати текстові дані у числові, але й тренувати модель для різного роду класифікаційних задач і т.д.

Отже, за допомогою кодування текстових даних таким способом можна значно зменшити обчислювальні ресурси апаратного забезпечення, які потрібні для реалізації комп’ютерної системи.

2.2 Огляд архітектур типу Transformers

Transformers - це модель глибокого навчання, запроваджена в 2017 році, що використовується переважно у галузі обробки природної мови (NLP). Як і рекурентні нейронні мережі (РНН), Трансформери призначені для обробки послідовностей даних природної мови, для таких завдань, як переклад та узагальнення тексту. Однак, на відміну від RNN, Трансформери не вимагають обробки послідовних даних у визначеному порядку. Наприклад, якщо вхідними даними є речення природної мови, то моделі не потрібно обробляти лише його початок протягом усього терміну обчислення. Завдяки цій особливості, Трансформери дозволяють значно більше розпаралелювати процес опрацювання вхідних даних, ніж РНН, а отже, і пришвидшити процес навчання.

Трансформери швидко стали найпоширенішою архітектурою для вирішень задач у сфері обробки природної мови, замінивши старі рекурентні моделі нейронних мереж, такі як довга короткочасна пам'ять (ДКЧП). Оскільки модель Transformers сприяє більшому розпаралеленню під час навчання, вона дозволила тренуватися на об’ємніших наборах даних, ніж це було можливо раніше. Це призвело до розробки попередньо навчених систем, таких як BERT (Представлення двонаправлених кодерів від Трансформерів) та GPT (Генеративний попередньо підготовлений Трансформер), які пройшли навчання з величезними загальними мовними наборами даних і можуть бути точно налаштовані на конкретні мовні завдання.

2.2.1 Реалізація архітектури Transformers для вирішення проблем задач типу “послідовність-послідовність”. Для програмної реалізації архітектури типу Transformers потрібно імплементувати три основних компоненти останньої: енкодер, декодер, замаскований шар Multi-Head Attention.

Енкодер у моделях даного типу виглядає практично так само як і у попередніх реалізаціях Attention Mechanism-based підходу.

```
class PositionalEncoder(nn.Module):
    def __init__(self, d_model, max_seq_len = 80):
        super().__init__()
        self.d_model = d_model

        # create constant 'pe' matrix with values dependant on
        # pos and i
        pe = torch.zeros(max_seq_len, d_model)
        for pos in range(max_seq_len):
            for i in range(0, d_model, 2):
                pe[pos, i] = \
                math.sin(pos / (10000 ** ((2 * i)/d_model)))
                pe[pos, i + 1] = \
                math.cos(pos / (10000 ** ((2 * (i + 1))/d_model)))

        pe = pe.unsqueeze(0)
        self.register_buffer('pe', pe)


    def forward(self, x):
        # make embeddings relatively larger
        x = x * math.sqrt(self.d_model)
        #add constant to embedding
        seq_len = x.size(1)
        x = x + Variable(self.pe[:,:seq_len], \
        requires_grad=False).cuda()
        return x
```

Рис. 2.3 - Код позиційного енкодера моделі Transformers

Після створення позиційного енкодера потрібно реалізувати механізм Multi-Head Attention, зображений на рис. 2.3.

V, K and Q відповідають ‘key’, ‘value’ and ‘query’ (ключ-значення-запит). Це терміни, що використовуються у функціях уваги (self Attention), У випадку з енкодером, V, K і Q будуть просто ідентичними копіями вектора вбудовування (плюс позиційне кодування). Вони матимуть розмір batch_size * seq_len * d_model. Де

batch_size - розмір вікна, яке буде містити вхідні дані, seq_len - довжина послідовності, d_model - розмірність моделі (у тому числі кількість її шарів).

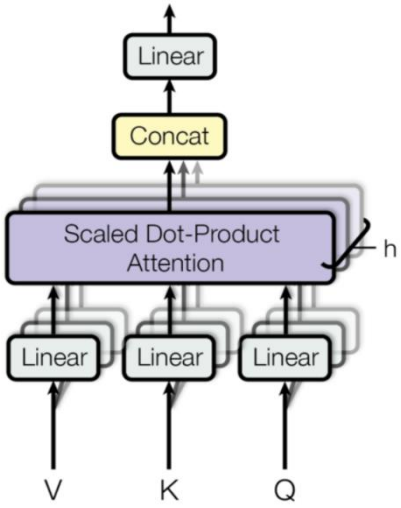

Рис. 2.4 - Механізм Multi-Head Attention

Реалізація механізму Multi-Head Attention потребує врахування усіх вищезгаданих параметрів для того, щоб процес “запам’ятовування” відбувався паралельно.

```python
class MultiHeadAttention(nn.Module):
    def __init__(self, heads, d_model, dropout = 0.1):
        super().__init__()

        self.d_model = d_model
        self.d_k = d_model // heads
        self.h = heads

        self.q_linear = nn.Linear(d_model, d_model)
        self.v_linear = nn.Linear(d_model, d_model)
        self.k_linear = nn.Linear(d_model, d_model)
        self.dropout = nn.Dropout(dropout)
        self.out = nn.Linear(d_model, d_model)

    def forward(self, q, k, v, mask=None):

        bs = q.size(0)

        k = self.k_linear(k).view(bs, -1, self.h, self.d_k)
        q = self.q_linear(q).view(bs, -1, self.h, self.d_k)
        v = self.v_linear(v).view(bs, -1, self.h, self.d_k)

        k = k.transpose(1,2)
        q = q.transpose(1,2)
        v = v.transpose(1,2)
        scores = attention(q, k, v, self.d_k, mask, self.dropout)

        concat = scores.transpose(1,2).contiguous()\
        .view(bs, -1, self.d_model)

        output = self.out(concat)

        return output
```

Рис. 2.5 - Реалізація механізму Multi-Head Attention

Схематично зобразити обчислення Multi-Head Attention можна наступним чином:

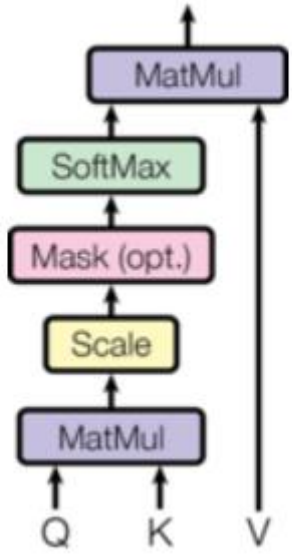

Рис. 2.6 - Схема обчислення Mult-Head Attention

На вищенаведеній схемі зображено чотири основних елемента для реалізації механізму “самозапам’ятовування”, отже:

1. MatMul - функція матричного множення двох тензорів.
2. SoftMax - функція приведення заданих значень до необхідного проміжку.
3. Mask - функція приховування частини навчальних даних (використовується для запобігання процесу перенавчання нейронної мережі).
4. Scale - функція масштабування вхідних даних до потрібного проміжку.

Програмна реалізація вищенаведеної схеми потребує роботи з тензорами даними (багатовимірними матрицями), таким чином для швидкого обчислення операцій між ними ми будемо використовувати бібліотеку torch, яка містить у собі вже реалізовані функції множення матриць із використанням апаратного підсилення графічного ядра. Таким чином, на зображенні 2.7 наведено імплементування схеми механізму обчислення Multi-Head Attention.

```
def attention(q, k, v, d_k, mask=None, dropout=None):

    scores = torch.matmul(q, k.transpose(-2, -1)) /  math.sqrt(d_k)
if mask is not None:
        mask = mask.unsqueeze(1)
        scores = scores.masked_fill(mask == 0, -1e9)
scores = F.softmax(scores, dim=-1)

    if dropout is not None:
        scores = dropout(scores)

    output = torch.matmul(scores, v)
    return output
```

Рис. 2.7 - Код обчислення Multi-Head Attention

Наступним етапом реалізації архітектури Transformers є створення класичної нейронної мережі, структура якої була наведена у попередніх розділах.

```python
class FeedForward(nn.Module):
    def __init__(self, d_model, d_ff=2048, dropout = 0.1):
        super().__init__()
        # We set d_ff as a default to 2048
        self.linear_1 = nn.Linear(d_model, d_ff)
        self.dropout = nn.Dropout(dropout)
        self.linear_2 = nn.Linear(d_ff, d_model)
    def forward(self, x):
        x = self.dropout(F.relu(self.linear_1(x)))
        x = self.linear_2(x)
        return x
```

Рис. 2.8 - Реалізація класичної нейронної мережі

Для кращого розуміння роботи класичної нейронної мережі варто розібратись із основними складовими останньої та їх призначенням.

Отже, кожен нейрон у мережі здатний приймати вхідні сигнали, обробляти їх і відправляти вихідний сигнал. Також кожен нейрон пов'язаний принаймні із ще одним нейроном, і кожне з'єднання оцінюється дійсним числом, яке називається ваговим коефіцієнтом, що відображає ступінь важливості даного зв’язку в нейронній мережі.

Загалом, нейронна мережа має здатність апроксимувати, тобто реалізовувати довільне відображення одного векторного простору на інший. Головною перевагою нейронних мереж є той факт, що вони здатні використовувати деяку апріорі невідому інформацію, приховану в даних, так звані їх особливості. Процес «захоплення» невідомої інформації називається навчанням нейронної мережі.

У математиці, формально “вчитися” означає коригувати вагові коефіцієнти таким чином, щоб виконувались певні умови. Існує два основних типи навчального процесу: навчання під наглядом та без нагляду (із вчителем та без). Навчання з вчителем означає, що нейронна мережа знає бажаний вихід і коригування вагових коефіцієнтів зроблено таким чином, що розрахований і бажаний виходи максимально наближені. Навчання без учителя означає, що бажаний результат невідомий, система забезпечила групу фактів (закономірностей), а потім вже почала процес тренування - досягнення виокремлених результатів.

Звичайна нейронна мережа складається хоча б із трьох шарів: вхідний, прихований і вихідний. У вхідному шарі кількість нейронів відповідає розмірності тренувальних даних, прихований шар може містити довільну кількість нейронів (зазвичай значно більшу ніж у вхідному шарі), у вихідному шарі кількість нейронів відповідає числу розмірності очікуваних даних. Наприклад, для задачі бінарної класифікації будь-чого, вихідний шар матиме лише два нейрона.

Кожен нейрон відповідає за обчислення лінійної нерівності, яка виглядає наступним чином:

$$y = X \times W^T + b \tag{2.1}$$

де y - очікуваний результат на виході нейрона, X - це вхідні дані, W^T - ваги, які використовуються для навчання (початково вони задаються набором випадкових чисел), b - коефіцієнт зміщення (перший раз теж задається послідовністю випадкових чисел).

Останнім етапом створення архітектури типу Transformers є імплементування класу останніх для подальшого функціонального використання у розділі задач “послідовність-послідовність”.

```
class Transformer(nn.Module):
    def __init__(self, src_vocab, trg_vocab, d_model, N, heads):
        super().__init__()
        self.encoder = Encoder(src_vocab, d_model, N, heads)
        self.decoder = Decoder(trg_vocab, d_model, N, heads)
        self.out = nn.Linear(d_model, trg_vocab)
    def forward(self, src, trg, src_mask, trg_mask):
        e_outputs = self.encoder(src, src_mask)
        d_output = self.decoder(trg, e_outputs, src_mask, trg_mask)
        output = self.out(d_output)
        return output
```

Рис. 2.10 - Імплементування класу Transformers

Після з’єднання усіх блоків коду ми отримаємо повноцінну архітектуру Transformers. У нашому випадку її застосуванням буде полягати у вирішенні задачі нейронного машинного перекладу.

```
d_model = 512
heads = 8
N = 6
src_vocab = len(EN_TEXT.vocab)
trg_vocab = len(FR_TEXT.vocab)
model = Transformer(src_vocab, trg_vocab, d_model, N, heads)
for p in model.parameters():
    if p.dim() > 1:
        nn.init.xavier_uniform_(p)
optim = torch.optim.Adam(model.parameters(),
                         lr=0.0001,
                         betas=(0.9, 0.98),
                         eps=1e-9)
```

Рис. 2.11 - Приклад коду для тренування моделі Transformers

Для остаточного процесу тренування моделі потрібно правильно підготувати тренувальні дані, якість і кількість яких будуть визначати якість перекладу. У наступному розділі буде розглянуто практичну реалізацію процесу підготування тренувальних даних, підготування моделі до тренування, тренування та оцінки якості отриманого перекладу.

РОЗДІЛ 3
РЕАЛІЗАЦІЯ СИСТЕМИ ПЕРЕКЛАДУ ПРИРОДНОЇ МОВИ НА ОСНОВІ МОДЕЛІ "ПОСЛІДОВНІСТЬ-ПОСЛІДОВНІСТЬ" ТА НЕЙРОМЕРЕЖЕВОЇ АРХІТЕКТУРИ ТРАСНФОРМЕРС

У цьому розділі наведено програмну реалізацію основних етапів створення системи перекладу, зокрема - підготовка тренувальної та тестувальної вибірок даних, підготовка архітектури Transformers до тренування, тренування моделі глибокого машинного навчання, оцінка якості натренованої моделі, розгортання програмного рішення на комп'ютерних системах різних типів за допомогою платформи Docker.

Для реалізації системи перекладу природної мови на основі нейромережевої моделі глибокого машинного навчання потрібно спроектувати комп'ютерну систему, яка б задовільняла усіх апаратним вимогам для тренування нейронної мережі. Зокрема, нижче наведено таблицю з усіма необхідними апаратними компонентами та їх вартістю.

Таблиця 3.1

Перелік апаратних комплектуючих комп'ютерної системи

Тип	Назва	К-сть	Вартість
Материнська плата	MSI B450-A Pro Max Socket AM4	1 шт.	100 $
Відеокарта	EVGA GeForce GTX 1080 Ti SC Black Edition Gaming, 11GB GDDR5X	4 шт.	4 x 1000 $
Процесор	AMD Ryzen 3 3100	1 шт.	140 $
ОЗП	DDR4 16GB/3200 Team Elite	2 шт.	2 x 55 $
Системний блок	GameMax ET-212-NP-U3	1 шт.	20 $
Блок живлення	Asus ROG Thor 1200W	1 шт.	350 $
	Загальна вартість:	-	4720 $

Опціональним рішенням для проектування даної комп'ютерної системи є водяне охолодження центрального процесора материнської плати та графічних ядер

відеокарт. Для тренування моделі на помірній кількості даних (приблизно до 2 мільйонів стрічок) додаткового охолодження не потрібно.

3.1 Створення векторизатора для завантажених даних

Першим етапом у виконанні поставленого завдання є підбір правильних моделей і бібліотек для їх подальшого використання, нижче наведений перелік деяких бібліотек і модулів, які потрібні для реалізації проекту:

```
import tensorflow as tf
tf.logging.set_verbosity(tf.logging.ERROR
import tensorflow_hub as hub
import numpy as np
from nltk.corpus import wordnet as wn
import csv

import re
import pickle
from nltk.stem import SnowballStemmer
from nerprocess import preprocess
s_stemmer = SnowballStemmer("english")

ner = preprocess()
embed = hub.Module(
    "https://tfhub.dev/google/universal-sentence-encoder-large/3"
    )

```

Рис. 3.1 - Імпортування необхідних бібліотек та ініціалізація параметрів моделі

Модель векторного представлення USE написана за допомогою фреймворку TensorFlow, адже саме він дає змогу використовувати апаратні ресурси комп’ютера найбільш правильно, виконуючи операції з тензорами на графічному відеоядрі, а не на процесорному. Таким чином можна досягнути значного приросту швидкодії. Наступним етапом є імпортування збірки усіх попередньо натренованих моделей TensorFlow для подальшого вибору необхідної. Як вже зазначалось вище, для обробки та приведення тексту до нормальної форми, у сфері задач NLP найчастіше використовують дві бібліотеки - NLTK і GenSim. Зокрема для стемінгу слів, тобто приведення до основної форми, було обрано використати SnowBall Stemmer, адже

він зарекомендував себе найкраще на українській мові. Опис використання утиліти flair для тегування NER буде наведено згодом.

Ключовим моментом є виконання роботи безпосередньо моделі USE, нижче наведений фрагмент коду, мета якого завантажити модель для її подальшого використання:

```
def generate_features(self, texts):
        """Creates 512 embedding
        for input string
        """
        if type(texts) is str:
            texts = [texts]

        g = tf.Graph()
        with g.as_default():
            text_input = tf.placeholder(dtype=tf.string, shape=[None])
            embed = hub.Module(
                "https://tfhub.dev/google/universal-sentence-encoder-large/3"
                )
            my_result = embed(text_input)
            init_op = tf.group([
                tf.global_variables_initializer(),
                tf.tables_initializer()])
        g.finalize()

        with tf.Session(config = tf.ConfigProto(device_count = {'GPU': 1})) as sess:
            with tf.Session() as sess:
                sess.run([
                    tf.global_variables_initializer(),
                    tf.tables_initializer()])
                return sess.run(embed(texts))

        # Create session and initialize.
        session = tf.Session(graph=g)
        session.run(init_op)
        return session.run(my_result, feed_dict={text_input: texts})
```

Рис. 3.2 - Код завантаження моделі та ініціалізації графа TF

Для високої якості подальшого порівняння векторів між собою потрібно забезпечити достатню векторну розмірність, у даному випадку вона рівна 512. Насправді - це досить велике число і з його опрацюванням справиться далеко не кожен комп’ютер, проте, оскільки модель попередньо натренована компанією Google, можливість використання 512 розмірного вектора стала цілком реальною.

3.2 Обробка даних та визначення міри семантичної наближеності векторів

Наступними кроками буде попереднє опрацювання вхідного тексту, аби в подальшому його можна було подати на вхід моделі, нижче наведено основні функції опрацювання текстових даних і пояснення до них:

```
def remove_str(self, text, min_len, max_len):
    """Removes strings that do not
    match desired amount of tokens
    """
    text = text.split(" ")
    if len(text) < min_len or len(text) > max_len:
    text = [""]
    return " ".join(text)
```

Рис. 3.3 - Код для фільтрування вхідного тексту по кількості слів

Вищенаведений код відсікає усі стрічки, кількість токенів яких більша або менша заданого числа(передається гіперпараметром при запуску моделі, або використовуються значення 2 і 10). Після відбору лише потрібних стрічок, функція повертає список із вже готових до подальшого опрацювання наборів токенів.

```
def process_data(self):
    data = self. data
    min_len = self.min_len
    max_len = self.max_len

    data = [self.process_text(text) for text in data]
    data = [self.remove_str(text, min_len, max_len) for text in data]
    while("" in data) :
    data.remove("")

    data_ = []

    for i in range(len(data)):
    data_.append((data[i], [i]))
    self.data_ = data_

    return data
```

Рис. 3.4 - Імплементування функції для токенізації вхідного тексту

Як видно, з коду наведеного вище, потрібно також вилучити усі пусті токени, щоби при перетворенні тексту у вектори не було зайвих або їх також називають неінформативних термів.

Подальшим етапом приготування текстових даних для їх подання в модель є перевірка на антоніми, дана перевірка зможе відсікти чимало апріорі семантично не подібних слів. Реалізація функції перевірки на антоніми наведено нижче:

```
ant1 = []
for token in tokens1:
    for ss in wn.synsets(token):
        syn_for_ant = []
    if ss.similar_tos():
for ssl in ss.similar_tos():
    syn_for_ant.append(ssl.lemmas())
    syn_for_ant = [x for sublist in syn_for_ant for x in sublist]
for lemma in ss.lemmas() + syn_for_ant:
    if lemma.antonyms():
        ant1.append(lemma.antonyms()[0].name())
        ant1 = [s_stemmer.stem(token) for token in ant1]
        ant2 = list()
for token in tokens2:
    if s_stemmer.stem(token) in ant1:
    return False
for ss in wn.synsets(token):
    syn_for_ant = []
    if ss.similar_tos():
        for ssl in ss.similar_tos():
            syn_for_ant.append(ssl.lemmas())
            syn_for_ant = [x for sublist in syn_for_ant for x in sublist]
for lemma in ss.lemmas() + syn_for_ant:
    if lemma.antonyms():
        ant2.append(lemma.antonyms()[0].name())
        ant2 = [s_stemmer.stem(token) for token in ant2]
for token in tokens1:
    if s_stemmer.stem(token) in ant2:
    return False
return True
```

Рис. 3.5 - Реалізація функції для пошуку антонімів

Для визначення антонімів між словами потрібно провести звичайну перевірку їх тегів у згаданій раніше мережі слів WordNet. Після проведення усіх вищенаведених етапів опрацювання тексту можна подавати текстовий корпус на вхід моделі, щоби потім виконати порівняння векторів.

```python
def cosine_similarity(self, v1, v2):
    """Calculates cosine similarity between
    vector representations of two strings
    """
    mag1 = np.linalg.norm(v1)
    mag2 = np.linalg.norm(v2)
    if (not mag1) or (not mag2):
        return 0
    return np.divide(np.dot(v1, v2), (mag1 * mag2))
```

Рис. 3.6 - Код функції пошуку кута косинуса між двома векторами

Коли текст представлено у векторній формі, можна використати формулу кута між двома векторами, величина результату буде вказувати на ймовірність семантичної подібності обидвох слів. Чим менший кут між двома векторами, тим більша ймовірність того, що слова мають смислову схожість і, відповідно, навпаки, чим більший кут між векторами, тим дальші за своїм сенсом два слова.

```python
def search_similar(self, query, threshold = 0.9):
    """Takes query string and threshold as an input
    Calculates cosine distance between vector representation
    of query string and each previously generated embedding.
    Returns array of tuples that hold (cosine distance, string, string id)
    """

    query = self.process_text(query)
    query_vec = self.generate_features(query)[0].ravel()
    res = []
    for i, d in enumerate(self.data):
        qvec = self.vectors[i].ravel()
        sim = self.cosine_similarity(query_vec, qvec)
    if sim > threshold:
        if self.not_antonyms(query, d):
            res.append((sim, d[:100], i))

    return sorted(res, key=lambda x : x[0], reverse=True)
```

Рис. 3.7 - Код функції знаходження семантично подібних речень

Даний фрагмент коду відповідає за порівняння векторів між собою та подальшим передаванням результату у подальші функції, параметр threshold вказує значення ймовірності, нижче якого результати не будуть братись до уваги, часто можна зустріти його під назвою “поріг”.

Після порівняння та відсікання нерелевантних векторів потрібно виконати їх сортування. Зазначимо, що було випробувано три різним методів сортування та групування результатів, зокрема:

– стандартне сортування за спаданням ймовірності подібності;

– групування методом транзитивності(якщо стрічка A подібна до стрічки B, а стрічка B подібна до C, то стрічки A і C - семантично схожі);

– групування з усуненням поточної стрічки з інших груп семантично подібних стрічок.

Останнім етапом розробки інструменту для пошуку семантичних дублікатів є запис результатів до файлу, для зручності подальшої роботи з результатами було обрано записувати всі дані у CSV(comma separated values) формат.

```
with open(output, 'w') as csvfile:
    pass
with open(output, 'w') as csvfile:
    filewriter = csv.writer(csvfile, delimiter='\n',
    quotechar='|', quoting=csv.QUOTE_MINIMAL)
for group in sorted_output:
    filewriter.writerow(["\" " + str(string[2]) + " \" , \" " +
    str(string[0]) + " \" , " + str(string[1]) for string in group])

    filewriter.writerow([])
```

Рис. 3.8 - Запис отриманих результатів у файл

Результат роботи програми для пошуку семантичних дублікатів зображено на рисунку 3.9, для прикладу було обрано текстовий корпус відкритих ресурсів програми Telegram:

	A	B	
1	[3]	" Invalid Country Code "	o
2	[1284]	" Country code not found "	0.949
3			
4	[853]	" Code expired	o
5	[1224]	" Invalid code. Please try again. "	0.876
6			
7	[14]	" Didn\'t get the code? "	o
8	[1287]	" Haven\'t received the code? "	0.931
9			
10	[787]	" The verification code was sent to your email. "	o
11	[1544]	" The verification code has been resent to your email. "	0.86
12			
13	[42]	" Your email address "	o
14	[770]	" Your Email Code "	0.894
15	[660]	" Enter your email address "	0.887
16			
17	[39]	" Password & Email "	o
18	[40]	" Enter a password "	0.867
19	[1220]	" Re-enter a password "	0.86
20	[1156]	" Confirm your password "	0.856
21			
22	[41]	" Re-enter your password "	o
23	[769]	" Enter your new password "	0.955
24	[1221]	" Enter a new password "	0.944
25	[768]	" Enter your password "	0.939
26	[1161]	" Enter your password to unlock "	0.902
27	[767]	" Set Additional Password "	0.873
28	[1218]	" Set additional password "	0.873
29	[773]	" Turn Password Off "	0.872
30	[719]	" Re-enter your new passcode "	0.86

Рис. 3.9 - Результат виконання програми для пошуку семантичних дублікатів

Як видно з наведеного вище зображення, у стовпці А містяться порядкові номери стрічок у оригінальному документі, В містить самі стрічки та С - ймовірність того, що стрічки семантично схожі. Саме такий підхід надасть нам змогу оцінити якість отриманого перекладу без витрат на людські ресурси.

3.3 Підготовка даних до тренування та тренування нейромережевої моделі глибокого машинного навчання Transformers

Для того, щоб натренувати модель глибокого машинного перекладу для задачі перекладу текстів природної мови, потрібно мати дані, які задовільнятимуть двом умовам:

1. Тренувальні дані повинні мати не дуже велику кількість слів в одному рядку (максимальна довжина рядка визначається апаратним забезпеченням ЕОМ, на якій буде відбуватись тренування).

2. Текст мови оригіналу та очікуваного виходу повинні бути паралельними, тобто кожен рядок має відповідати за змістом протилежному.

Першим етапом підготовки даних до тренувального процесу є їх чистка від зайвої інформації. На рисунку нижче наведено процес імпортування необхідних бібліотек.

```
import re
import numpy as np
import pandas as pd
from pandarallel import pandarallel

pandarallel.initialize()
```

```
INFO: Pandarallel will run on 16 workers.
INFO: Pandarallel will use Memory file system
```

Рис. 3.10 - Імпортування необхідних бібліотек

Наступним етапом є читання вхідних даних, для їх подальшого опрацювання.

Read dictionary

```
[278]: dictionary = pd.read_csv("data/en-uk.dic", sep="\t", header=None)
       dictionary = dictionary[[dictionary.columns[3], dictionary.columns[2]]]
       dictionary.rename(columns={dictionary.columns[0]: "ua", dictionary.columns[1]: "en"}, inplace=True)

[279]: dictionary
```

[279]:

	ua	en
0	Кість	A bone of
1	Один	A certain
2	Добрий	A good
3	Трохи	A little while
4	Трошки	A little
...	...	...
25249	ревнителів	zealous
25250	же ревний	zealous
25251	дбаєте	zealous
25252	ревнителї	zealous
25253	Ревнують	zealously

25254 rows × 2 columns

Рис. 3.11 - Читання вхідних даних

Після читання вхідних даних, їх потрібно опрацювати та почистити від зайвої інформації. У нашому випадку цією інформацією виступають теги розподілу текстових даних, наприклад: <f>[2]</f>, </pb> і т.п. Для того, щоб забрати теги з даних, потрібно застосувати заміну останніх за допомогою регулярних виразів.

```
def get_cleaned_text(row: str, en: bool = True) -> str:
    """Function to get cleaned
    text from row with regex
    """
    if not isinstance(row, str):
        row = str(row)
    if en:
        row = re.sub(r'(<f>.{1,3}<\/f>)|(<pb/>)', '', row)
        row = re.sub(r'<.>|<\/.>', '', row)
    else:
        row = re.sub(r'<.>|<\/.>', '', row)
    row = re.sub(r'\[\d+\]', '', row)
    row = re.sub(r'\s+', ' ', row)

    return row.strip()
```

```
%%time

data["en"] = data["en"].apply(lambda x: get_cleaned_text(x, en=True))
data["ua"] = data["ua"].apply(get_cleaned_text)
```

```
CPU times: user 899 ms, sys: 0 ns, total: 899 ms
Wall time: 898 ms
```

Рис. 3.12 - Функція видалення непотрібних тегів

Наступним кроком є розбиття даних на тренувальну та тестувальну вибірки та запис останніх у відповідні файли.

```
#Write data to the files:
with open("ua-en.orig/tmp/train.ua", "w") as file:
    for line in train["ua"].to_list():
        file.writelines(line + "\n")

with open("ua-en.orig/tmp/train.en", "w") as file:
    for line in train["en"].to_list():
        file.writelines(line + "\n")

with open("ua-en.orig/tmp/test.ua", "w") as file:
    for line in test["ua"].to_list():
        file.writelines(line + "\n")

with open("ua-en.orig/tmp/test.en", "w") as file:
    for line in test["en"].to_list():
        file.writelines(line + "\n")
```

Рис. 3.13 - Запис тренувальної та тестувальної вибірок у відповідні файли

Після проходження усіх вищезгаданих етапів можна запускати процес тренування нейронної моделі.

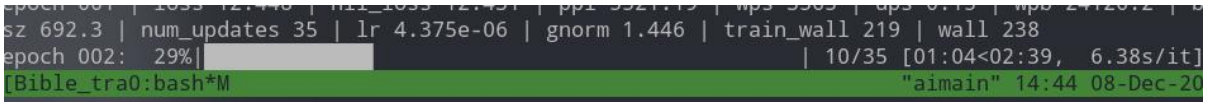

Рис. 3.14 - Процес тренування нейронної мережі

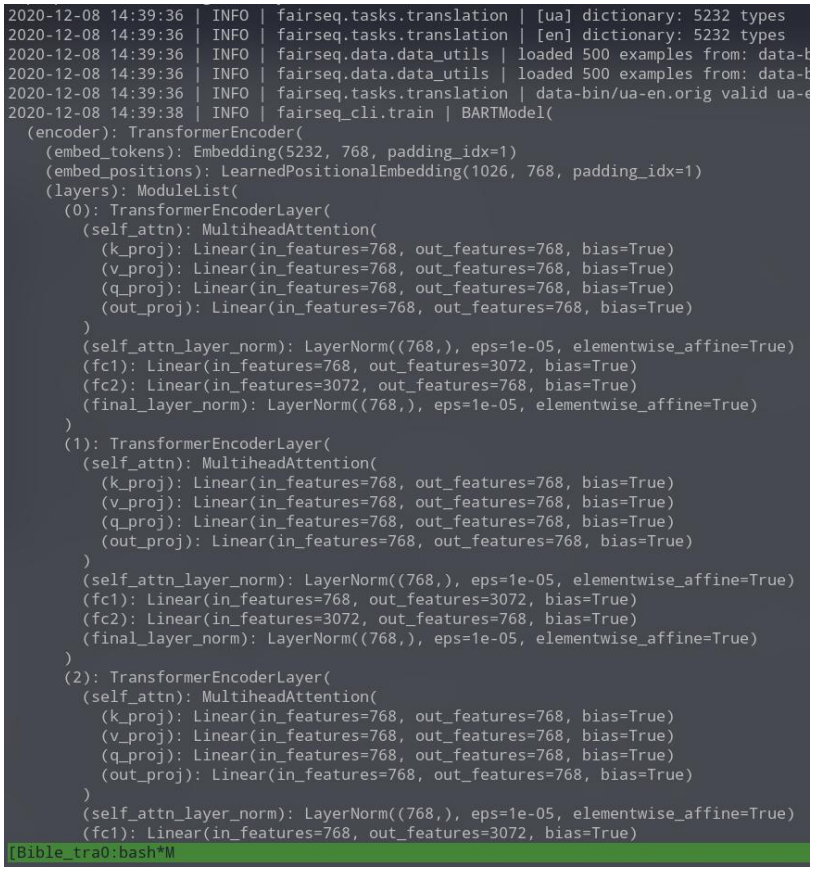

Рис. 3.15 - Структурний вигляд нейромережевої архітектури Transformers

У поточному випадку епоха - це 35 ітерацій, зокрема, після кожної пройденої епохи відбувається процес валідації даних, іншими словами - оцінки якості перекладу на даному етапі тренування.

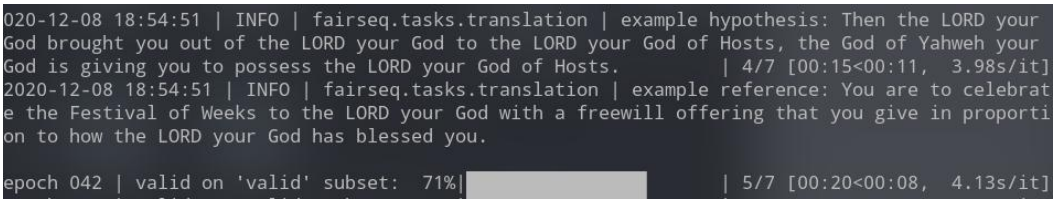

Рис. 3.16 - Процес оцінки якості перекладу

3.4 Розгортання універсального робочого середовища за допомогою платформ Docker та Docker-Compose

Після процесу тренування нейронної мережі та оцінки якості її перекладання, потрібно підготувати та розгорнути робоче середовище, за допомогою якого користувач міг би взаємодіяти із моделлю глибокого машинного навчання звичними для нього інструментами.

Саме тому для використання було обрано платформи Docker та Docker-Compose, адже саме дані технології здатні забезпечити максимальну кросплатформенність програмного рішення для використання на різноманітних ЕОМ.

Структура файлових директорій для використання технології Docker повинна мати наступний вигляд:

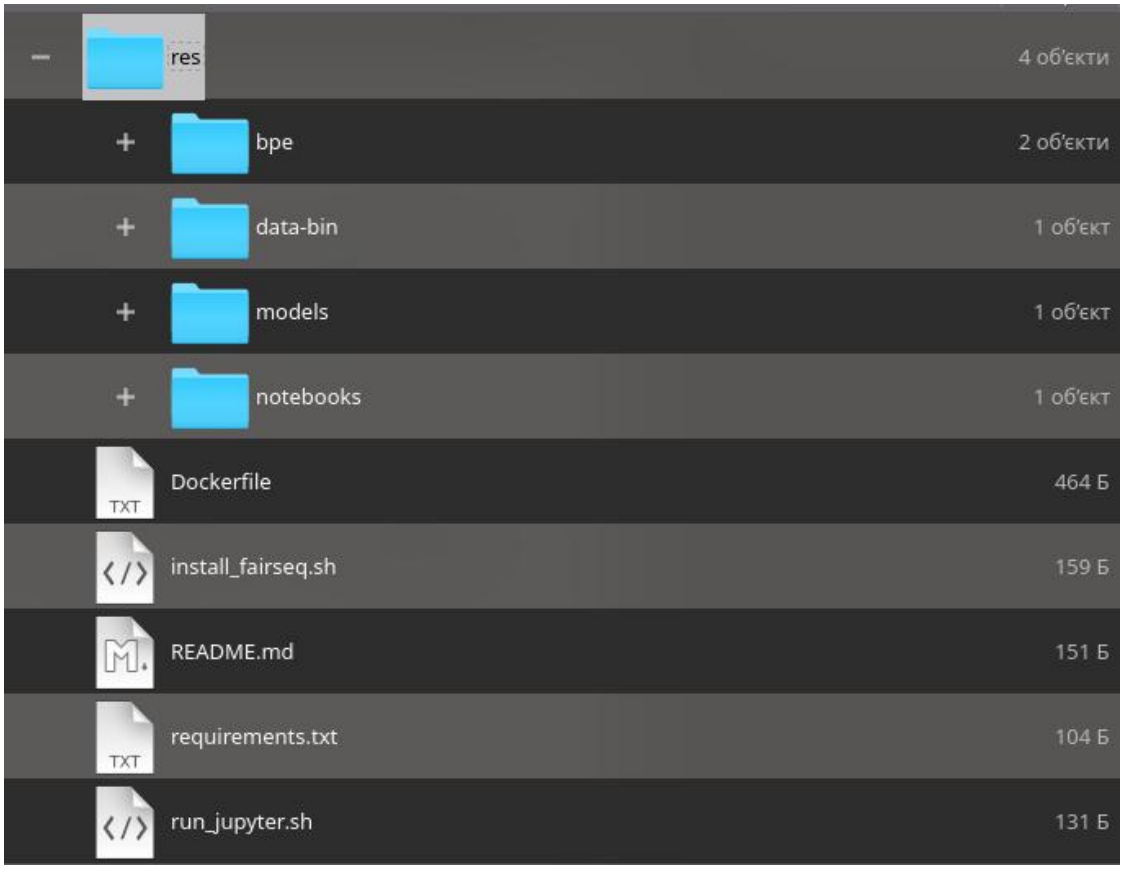

Рис. 3.17 - Вигляд структури файлових директорій

Для створення Docker образу потрібно прописати код конфігурації останнього у файлі з назвою Dockerfile, таким чином, для вищенаведеної структури файлових директорій код конфігурації повинен бути наступним:

```
FROM python:3.7-slim

MAINTAINER <Made by Roman Lutsyshyn

COPY . /home/

RUN apt update && apt-get install --reinstall build-essential -y && \
    apt-get install git -y && rm -rf /var/lib/apt/lists/* && \
    pip install --no-cache-dir -r /home/requirements.txt

WORKDIR /home/

RUN chmod +x /home/install_fairseq.sh && chmod +x /home/run_jupyter.sh && \
    bash install_fairseq.sh

CMD /home/run_jupyter.sh

```

Рис. 3.18 - Файл конфігурації Docker образу

Наступним етапом є побудова образу та його запуску. Для побудови образу Docker потрібно перейти в термінал та використати команду docker build -t nmt . Після чого почнеться процес побудови. Далі потрібно запустити образ для того, щоб він у свою чергу запустив внутрішній контейнер із необхідними для нас даними та файлами. Для запуску поточного образу потрібно виконати в терміналі команду docker run nmt.

Після виконання усіх вищенаведених дій попередньо натренована нейромережева модель глибокого машинного навчання готова до експлуатації. Вихідним результатом роботи нейронної мережі є словник числових індексів усіх термінів тренувальних даних, а також бінарний файл ваг і станів моделі, за допомогою яких вона висуває свої гіпотези потенційного перекладу.

```
from fairseq.models.transformer import TransformerModel
import torch
```

```
ua2en = TransformerModel.from_pretrained(
    '/home/roma/big_storage/bible_nmt/checkpoints/',
      checkpoint_file='checkpoint_best.pt',
      data_name_or_path='/home/roma/big_storage/bible_nmt/data-bin/ua-en.orig/',
      bpe='fastbpe',
      bpe_codes='/home/roma/big_storage/bible_nmt/ua-en.orig/bpe/codes',
)
```

```
print(ua2en.translate("Він сподівається тільки стати вільним, але Бог спасе нас."))
```

```
He trusts only will become free, but God will preserve us.
```

```
print(ua2en.translate("Він сподівається тільки стати вільним, але Бог спасе нас."))
```

```
He lives only to become free, but God will preserve us.
```

Рис. 3.19 - Переклад довільного тексту за допомогою натренованої моделі

Отже, як видно із рисунка 3.19, процес тренування моделі пройшов успішно, оскільки отриманий переклад відповідає вхідному тексту. Для того, щоб виключити елемент людського фактору при перевірці якості отриманого перекладу було висунуто наступну гіпотезу: семантична подібність отриманого перекладу та перекладу з тестувального набору даних має бути дуже висока, в окремих випадках - тексти повинні бути ідентичними.

Реалізовувати таке рішення будемо за допомогою описаних раніше енкодерів текстових даних у векторний простір та подальшим обчислення кута косинуса між ними, який у нашому випадку буде прямо пропорційним ймовірності семантичної подібності двох текстових стрічок. Отримані результати вважаємо метриками поточного підходу.

```
#This func is for second approach --> using USE
def extract_embedding(sentence):
    '''
    Fucntion to get 512 dimension vector from each sentence
    '''
    embedding = session.run(embedded_text, feed_dict={text_input: [sentence]})
    dict_update = {i: embedding[0][i] for i in range(512)}

    return pd.Series(dict_update).values.reshape(1, -1)
```

```
extract_embedding("He trusts in only will become free, but God will preserve us.")
```

```
array([[ 0.05143499, -0.00693954,  0.02342568, -0.03532691,  0.05542584,
        -0.04637785,  0.08134337,  0.00286598, -0.04011842,  0.0594504 ,
         0.02577414,  0.00250251,  0.05199176,  0.06137231,  0.03400113,
         0.05945102,  0.03908326,  0.07853518,  0.07664886, -0.04089533,
         0.00739397,  0.03896992,  0.02141231, -0.03031405,  0.02637491,
         0.08566679, -0.02947787,  0.02502444,  0.03233388, -0.00772289,
        -0.00963265, -0.01104937,  0.01265907,  0.01492304, -0.01880576,
        -0.03789835,  0.08371437, -0.05651744, -0.00237513,  0.06174321,
         0.01448935, -0.04799851,  0.01281469,  0.0043236 ,  0.07553078,
         0.01623062,  0.03128868,  0.08099955,  0.09083031, -0.03243934,
        -0.03367953,  0.04836444, -0.01730898, -0.06510018,  0.03770936,
        -0.07105926,  0.06181644,  0.05121416, -0.02388687, -0.02478533,
        -0.06455599,  0.05230835, -0.056319  ,  0.0241478 , -0.05831901,
        -0.03089125, -0.0288578 ,  0.04093131, -0.04007378,  0.01053239,
         0.01641571, -0.01594106,  0.04873278,  0.09706581,  0.00380916,
        -0.0449598 ,  0.03509548, -0.06407363,  0.03750606,  0.00153441,
        -0.00734204,  0.05865727,  0.01429237, -0.04344694,  0.01014578,
         0.05699039,  0.0865151 ,  0.04249965,  0.04151209,  0.01831226,
         0.05393093,  0.00164886,  0.02662269,  0.02045471,  0.04046808,
        -0.01649177,  0.00084169,  0.00293879, -0.01414838,  0.01595731,
        -0.065769  , -0.00128139, -0.00781075, -0.08285961, -0.07274919.
```

Рис. 3.20 - Вигляд векторизованої стрічки

Після обчислення косинуса кута між двома векторами, ми отримаємо значення в проміжку від 0 до 1, де 0 - семантично не схожі стрічки, а 1 - семантично ідентичні. На рисунку XX наведено результат семантичного порівняння отриманого перекладу від моделі та очікуваної стрічки з тестувального набору даних.

```
from sklearn.metrics.pairwise import cosine_similarity
```

```
cosine_similarity(extract_embedding("He trusts in only will become free, but God will preserve us."),
                  extract_embedding("He relies only on being free, but God will save us"))[0][0]
```

```
0.8796609201996296
```

Рис. 3.21 - Аналітичне зображення семантичної схожості текстових даних

У даному розділі було реалізовано комп’ютерну систему та розгорнуто на ній програмне забезпечення лдя вирішення задачі перекладу природної мови за допомогою нейромережевої моделі Transformers.

РОЗДІЛ 4
ОХОРОНА ПРАЦІ ТА БЕЗПЕКА В НАДЗВИЧАЙНИХ СИТУАЦІЯХ

4.1 Охорона праці

Розробка будь-якого програмного забезпечення передбачає його подальше розгортання та використання в різного роду комп’ютерних системах. Саме тому така система повинна відповідати всім нормам та правилам техніки безпеки, зокрема, стандартам і правилам із охорони праці. Дотримання останніх є рівнозначно важливим як для розробника та людини, яка проектує систему, так і для її кінцевого користувача.

Сучасний розвиток технічного та технологічного стану виробництва передбачає постійну автоматизацію та оптимізацію виробничих процесів. На сьогоднішній день, напевно, важко уявити підприємства, будь-яка діяльність в яких здійснювалась би без використання комп’ютерної техніки. Через масовий характер робіт, що виконуються працівниками за допомогою комп’ютера, законодавством України чітко врегульовано норми та вимоги до використання комп’ютерної техніки на підприємстві. Також на робочих місцях працівників необхідно забезпечити дотримання вимог, затверджених Наказом Мінсоцполітики від 14.02.2018 за № 207 «Про затвердження Вимог щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями».

Згідно Вимог приміщення, де розміщені робочі місця операторів, крім приміщень, у яких розміщені робочі місця операторів великих ЕОМ загального призначення (сервер), мають бути оснащені системою автоматичної пожежної сигналізації відповідно до цих вимог:

– переліку однотипних за призначенням об'єктів, які підлягають обладнанню автоматичними установками пожежогасіння та пожежної сигналізації, затвердженого наказом Міністерства України з питань надзвичайних ситуацій та у справах захисту населення від наслідків Чорнобильської катастрофи від 22.08.2005 N 161, зареєстрованого в Міністерстві юстиції України 05.09.2005 за N 990/11270 (НАПБ

Б.06.004-2005);

– Державних будівельних норм "Інженерне обладнання будинків і споруд. Пожежна автоматика будинків і споруд", затверджених наказом Держбуду України від 28.10.98 N 247 (далі - ДБН В.2.5-56:2014, з димовими пожежними сповіщувачами та переносними вуглекислотними вогнегасниками.

В інших приміщеннях допускається встановлювати теплові пожежні сповіщувачі. Приміщення, де розміщені робочі місця операторів, мають бути оснащені вогнегасниками, кількість яких визначається згідно з вимогами ДСТУ 4297:2004 «Пожежна техніка. Технічне обслуговування вогнегасників». Загальні технічні вимоги і з урахуванням граничнодопустимих концентрацій вогнегасної рідини відповідно до вимог НАПБ А.01.001-2014

4.1.1 Загальні вимоги щодо охорони праці при роботі з комп'ютером. Приміщення, в яких планується установка та подальша робота з комп'ютером, повинні відповідати проектній документації будинку, погодженій з уповноваженими державними органами. Крім того, роботодавець повинен враховувати існуючі санітарні нормативи освітлення, вимоги до параметрів мікроклімату (температура, відносна вологість), ступеня і сили вібрації, звукового шуму і вогнестійкості приміщення, а також характеристики електромагнітного, ультрафіолетового та інфрачервоного полів. Конкретні показники зазначених санітарних норм вказані у Державних санітарних правилах і нормах.

Так, наприклад, роботодавцю заборонено встановлювати комп’ютери в приміщеннях, розташованих у підвалах будинків та у цокольному поверсі. Приміщення де встановлені відеодисплейні термінали повинні бути укомплектовані системами центрального або індивідуального опалення, кондиціонування чи вентиляції повітря. Але при установці зазначених систем, необхідно переконатись, що батареї опалення, водопровідні труби, вентиляційні кабелі тощо, надійно сховані під захисними щитками, які перешкоджатимуть можливому потраплянню робітника під напругу.

У кожній кімнаті, де обладнуватимуться робочі місця співробітників, що

працюватимуть на комп’ютері, повинні бути наявні елементи природного та штучного освітлення. При цьому, на вікнах слід встановити легко регульовані жалюзі чи штори, які дозволять працівникам коригувати рівень освітлення в приміщенні. Робочі місця з відеодисплейними терміналами слід так розташовувати відносно світлових прорізів, щоб природне світло падало збоку, переважно зліва.

З метою досягнення максимального рівня безпечності і охорони праці при роботі з комп’ютером, виробничі приміщення необхідно обладнати аптечками першої медичної допомоги, системами автоматичної пожежної сигналізації і вогнегасниками. У приміщенні, де одночасно експлуатуються понад п'ять комп’ютерів, тощо, на помітному та доступному місці встановлюється аварійний резервний вимикач, який може повністю вимкнути електричне живлення приміщення, крім освітлення.

4.1.2 Вимоги до особистого робочого місця працівника. У кожній кімнаті, де обладнуватимуться робочі місця співробітників, що працюватимуть на комп’ютері, повинні бути наявні елементи природного та штучного освітлення. При цьому, на вікнах слід встановити легко регульовані жалюзі чи штори, які дозволять працівникам коригувати рівень освітлення в приміщенні. Робочі місця з відеодисплейними терміналами слід так розташовувати відносно світлових прорізів, щоб природне світло падало збоку, переважно зліва. При розміщенні робочих столів з відеодисплейними терміналами слід дотримувати такі відстані між бічними поверхнями відеодисплейних терміналів - 1,2 м, відстань від тильної поверхні одного відеодисплейного термінала до екрана іншого відеодисплейного термінала - 2,5 м.

Конструкція робочого столу має відповідати сучасним вимогам ергономіки і забезпечувати оптимальне розміщення на робочій поверхні використовуваного обладнання (дисплея, клавіатури, принтера) і документів.

Робочий стілець має бути підйомно-поворотним, регульованим за висотою, з кутом і нахилу сидіння та спинки. Поверхня сидіння і спинки стільця має бути напів м'якою з нековзним, повітронепроникним покриттям, що легко чиститься і не

електризується. Площу та об'єм для одного робочого місця має бути не менше 6,0 кв.м, об'єм - не менше 20,0 куб.м.

На столі працівника можливо розмістити допоміжні для роботи пристрої (принтери, колонки, сканери), а також місця для зберігання документів, за умови, що це не обмежуватиме видимість екрану і не заважатиме працівнику. У разі надмірного шуму чи вібрації технічного обладнання, роботодавець повинен забезпечити працівників антивібраційними килимками.

Щодня необхідно проводити вологе прибирання приміщення, та очищати робоче місце та безпосередньо монітор комп’ютера від запиленості.

На підприємстві забороняється:

− проводити ремонт та технічне обслуговування комп’ютера за робочим місцем працівника;

− самостійно ремонтувати або намагатись здійснити технічне налагодження комп’ютера без залучення компетентних спеціалістів;

− складати на робочому місці зайві документи, деталі та предмети, що не потрібні для роботи;

− працювати на комп’ютері, у яких під час роботи з'являються нехарактерні сигнали, нестабільне зображення на екрані тощо;

− відключати захисні пристрої, самочинно проводити зміни у конструкції та складі комп’ютерів;

− допускати до роботи осіб, що в установленому порядку не пройшли інструктаж та перевірку знань з охорони праці, пожежної безпеки.

4.1.3 Вимоги до профілактичних медичних оглядів, режимів праці і відпочинку при роботі з комп’ютером. При прийнятті на роботу кожна особа має пройти медичний огляд. Окрім того, при подальшій трудовій діяльності на підприємстві, така особа підлягає регулярному медичному огляду не рідше ніж раз на 2 роки. Обов’язковим є проходження таких лікарів як терапевта, невропатолога та офтальмолога.

На підприємстві мають бути чітко встановлені перерви для відпочинку

працівників (окрім обідньої), як правило, тривалістю 10-15 хвилин раз на годину або дві, в залежності від складності роботи. В будь-якому випадку, роботодавець повинен передбачити такий розпорядок роботи на підприємстві, щоб час безперервної роботи з комп'ютером був не більше ніж 4 години.

Додатково, для збереження належного рівня здоров'я та професійної придатності робітників, рекомендується виділити на підприємстві окреме побутове приміщення для перепочинку працівників і зняття ними нервово-емоційного напруження, що виникає при роботі з комп'ютером.

Таким чином, даний підрозділ передбачає розгляд основних заходів, інструкцій та правил техніки безпеки при використанні комп'ютерних систем, зокрема, для вирішення задачі машинного перекладу текстів природної мови.

4.2 Безпека в надзвичайних ситуаціях

4.2.1 Вражаючі фактори надзвичайних ситуацій та оцінка стійкості підприємств до їх впливу. Стійкість роботи об'єкта народного господарства в НС оцінюємо за допомогою моделювання уразливості (характер руйнувань, пожеж, уражень робітників і службовців) об'єкта при впливі вражаючих факторів НС. Для цього використовуємо результати розрахункових даних. Основними вражаючими факторами НС є: повітряна ударна хвиля, світлове випромінювання, проникаюча радіація, радіоактивне зараження та електромагнітний імпульс. Всі ці вражаючі фактори можуть по різному впливати на функціонування об'єкту після НС. Тому оцінювати стійкість об'єкта потрібно по відношенню до кожного з вражаючих факторів. Також при НС виникають вторинні вражаючі фактори, які можуть значно впливати на роботу об'єкта (пожежі, вибухи, зараження отруйними і сильнодіючими отруйними речовинами місцевості, атмосфери і водойм, катастрофічне затоплення в зонах, розташованих нижче гребель гідровузлів, і т. п.). Це враховується при оцінці його стійкості. Оцінка стійкості до впливу повітряної ударної хвилі (ПУХ)

Мінімальне значення надлишкового тиску у фронті ВУВ, при якому будівлі, споруди та обладнання об'єкта отримують середні руйнування, - це кількісний

показник стійкості об'єкта до впливу повітряної ударної хвилі. Назвемо дане значення межею стійкості об'єкта до ударної хвилі і позначимо $\Delta P_{\Phi \lim}$ (КПа).

Оцінку стійкості проводимо в такій послідовності:

Спочатку визначаємо максимальне значення надлишкового тиску ПУХ, що впливає на об'єкт $\Delta P_{\Phi \max}$ (КПа):

- для ядерного вибуху (формула 4.1):

$$\Delta P_{\varphi max} = 95\frac{\sqrt[3]{G}}{R} + 390\frac{\sqrt[3]{G}}{R} + 1300\frac{G}{R^3} \qquad (4.1)$$

де G – потужність ядерного вибуху в тротиловому еквіваленті (ктн), R – відстань від центра вибуху (м),

- для вибуху газоповітряного середовища (формула 4.2):

$$\Delta P_{\varphi MAX} = \frac{700}{3\left(\sqrt{1+29{,}8 \bullet K^3} - 1\right)} \qquad (4.2)$$

де К – коефіцієнт, який виміряється по формулі (формула 4.3)

$$K = 0{,}24\frac{R}{17{,}5\sqrt[3]{G}} \qquad (4.3)$$

де R – відстань від центра вибуху (м),

G – маса вибухової речовини (т).

1. Виділяємо основні елементи на об'єкті, від яких залежить його робота.

2. Визначаємо ступені руйнування елементів об'єкта в залежності від повітряної ударної хвилі.

3. Потім визначаємо межу стійкості кожного окремого елемента (мінімальне значення надлишкового тиску, при якому елемент отримує середні руйнування).

4. Після того визначаємо межу стійкості об'єкта по мінімальній межі стійкості елементів, що входять до його складу $\Delta P_{\Phi \lim}$ (КПа) .

5. Висновок про стійкість об'єкта до впливу повітряної ударної хвилі такий: об'єкт стійкий,якщо $\Delta P_{Ф\lim}$ (КПа) $\geq \Delta P_{Ф\max}$ (КПа).

6. Висновки і пропозиції робимо на підставі аналізу результатів оцінки стійкості об'єкта, в яких вказується межа стійкості об'єкта, найбільш вразливі елементи, ступінь руйнування об'єкта.

Оцінка стійкості до дії теплового випромінювання. Показником стійкості об'єкта до впливу теплового випромінювання є мінімальне значення теплового імпульсу, при якому може статися запалення конструкцій будівель і виникнути пожежа. Це значення називають межею стійкості об'єкта до впливу теплового випромінювання та позначають $I_{Св\lim}$ (КДж /м2).

Оцінюємо стійкість так:

1. Насамперед визначаємо ступінь вогнестійкості будівель і споруд. Вогнестійкість будинку – здатність чинити опір впливу високих температур при збереженні своїх експлуатаційних властивостей. Вогнестійкість будинків залежить від меж вогнестійкості його основних конструктивних частин.

Межа вогнестійкості конструкції – це час у годинах, протягом якого конструкція виконує свої функції в умовах пожежі (тобто не згоряє, не тріскається, не деформується або поки температура на протилежній загорянню стороні не стане понад 140°С), залежить від поперечного перерізу, товщини захисного шару, займистості будівельних матеріалів (будівельні й інші матеріали бувають неспалимі, важко спалимі і спалимі), від здатності зберігати механічні властивості при впливі високих температур.

2. За ступенем вогнестійкості будинки і споруди поділяють на 5 груп:

I і II група – неспалимі; при загорянні предметів усередині будинку він охоплюється вогнем не раніше, ніж через 3–4 год.;

III група – неспалимі будинки зі спалимими перекриттями і перебірками; охоплюються вогнем через 2–3 год.;

IV група – дерев'яні, оштукатурені будинки; охоплюються вогнем через 1,5 год.;

V група – дерев'яні, неоштукатурені; охоплюються вогнем через 0,5 год.

3. Після того визначаємо в будівлях матеріали, що можуть горіти. 3. Далі знаходимо значення світлових імпульсів, при яких відбувається займання елементів, що найшвидше загоряються.

4. А в кінці визначаємо категорію виробництва за пожежною небезпекою.

Є такі категорії виробництва за пожежною небезпекою: А, Б, В, Г, Д, Е.

Категорія «А» (вибухонебезпечні виробництва) включає виробництва, які мають горючі гази з нижньою концентраційною межею загоряння в повітрі 10 % (об'ємних) і менше, рідини з температурою спалаху парів 28°С і нижче (при цьому гази і рідини можуть утворювати вибухонебезпечні суміші об'ємом, який перевищує 5 % об'єму повітря в приміщенні), а також речовини, здатні вибухати і горіти при взаємодії з водою, киснем повітря чи одна з одною. Це виробництва, де застосовуються металічні натрій і калій, ацетон, сірковуглець, ефір і спирти, а також фарбувальні цехи, об'єкти з наявністю зріджених газів.

Категорія «Б» – вибухопожежні виробництва, пов'язані із застосуванням горючих газів, нижня межа загоряння (НМЗ) яких понад 10 % до обсягу повітря, рідин з температурою спалаху від 28 до 61 °С включно; рідин, нагрітих в умовах виробництва до температури спалаху і вище; горючого пилу чи волокон, за умови, що ці гази, рідини і пил можуть утворити вибухонебезпечні суміші об'ємом, що перевищує 5 % об’єму приміщення. До цієї категорії відносяться насосні станції для перекачування рідин з температурою спалаху від 28 до 61 °С, виробництва с наявністю аміаку тощо.

Категорія «В» – пожежонебезпечні виробництва, пов'язані із застосуванням рідин з температурою спалаху парів вище 61 °С; горючого пилу чи волокон, речовин, здатних тільки горіти при взаємодії з водою, киснем чи одна з одною; твердих горючих речовин і матеріалів. До даної категорії відносяться виробництва по обробці деревини, торфу, вугілля, пластмас і гуми, склади горючих і мастильних матеріалів.

Категорія «Г» – виробництва, пов'язані з обробкою негорючих речовин і матеріалів у гарячому, розпеченому чи розплавленому стані, яка супроводжується виділенням променистого тепла, іскор і полум'я, твердих, 15 рідких і газоподібних

речовин, що спалюються чи утилізуються як паливо. До них відносяться цехи термообробки металу, газогенераторні станції, котельні. Категорія «Д» – виробництва, пов'язані з обробкою негорючих речовин і матеріалів у холодному стані. Це ділянки холодної обробки металів і т. п.

Категорія «Е» – вибухонебезпечні виробництва, пов'язані із застосуванням горючих газів без рідкої фази і вибухонебезпечного пилу у такій кількості, що вони можуть утворити вибухонебезпечні суміші об'ємом, що перевищує 5 % об'єму приміщення, у якому за умовами технологічного процесу можливий тільки вибух (без наступного горіння); речовин, здатних вибухати (без наступного горіння) при взаємодії з водою, киснем повітря чи одна з одною. До них відносяться ділянки електролізу води, зарядки і розрядки лужних і кислотних акумуляторів тощо.

5. Робиться висновок про стійкість об'єкта - об'єкт стійкий, якщо $I_{Cв\lim} \geq I_{Cв\max[}$.

Оцінка стійкості до впливу радіоактивного зараження

За показник стійкості об'єкта до впливу радіоактивного зараження взято граничне значення рівня радіації на об'єкті на початку радіоактивного зараження, при якому можлива виробнича діяльність у звичайному режимі (повними змінами, повний робочий день) і при цьому персонал не отримає дозу опромінення більше допустимої. Це значення вважається межею стійкості об'єкта до впливу радіації і позначається - $P_{1\lim}$ (рад / год).

Оцінка стійкості до впливу радіоактивного зараження проводиться в такому порядку:

1. Визначається ступінь захищеності персоналу або коефіцієнт ослаблення дози радіації будівлею і притулком за формулою (формула 4.3)

$$K_{ОСЛ.УБ.РЗ} = K_P \prod_{i=1}^{n} 2^{h_i/d_i} \qquad (4.3)$$

де Кр - коефіцієнт расположенья притулку;

hi - товщина і-того закисного шару;

di - товщина шару половинного ослаблення і-того захисного шару.

2. Визначається доза опромінення, яку отримає персонал. Доза випромінювання в умовах радіоактивного зараження в будівлі (формула 4.4)

$$Д_{ЗД.РЗ} = \frac{Д_{ОТКР}}{K_{ОСЛ.ЗД,РЗ}} = \frac{5P_1\left(t_н^{-0,2} - t_к^{-0,2}\right)}{K_{ОСЛ.ЗД.РЗ}} \quad (4.4)$$

де t н - час початку роботи в умовах радіоактивного зараження (РЗ), Р1 - максимальний рівень радіації на об'єкті.

3. Визначається межа стійкості об'єкта до впливу радіації, тобто граничне значення рівня радіації на об'єкті, до якого можлива робота в звичайному режимі (формула 4.5):

$$P_{1lim} = \frac{Д_{УСТ}K_{ОСЛ.ЗД.РЗ}}{5\left(t_н^{-0,2} - t_к^{-0,2}\right)} \quad (4.5)$$

ті електричних полів по формулам:

- вертикальної складової:

$$E\,г = 10\frac{1+2R}{R^3}lg14{,}5q \quad (4.4)$$

- горизонтальної складової:

$$E\,в = 5 \bullet 10^3\frac{1+2R}{R^3}lg14{,}5q \quad (4.5)$$

де R – відстань від центра вибуху (км),

q- потужність ядерного вибуху(ктн).

У даному розділі розглянуто способи та методи оцінки стійкості підприємств до різного роду вражаючих факторів. Зокрема, наведено класифікацію різноманітних об’єктів за ступенями вогнестійкості та межами стійкості до впливу радіації.

ВИСНОВКИ

Під час проведення даного наукового дослідження було пройдено три основних етапи: аналіз існуючих методів для вирішення задачі перекладу текстів природної мови, реалізація найкращого методу та вдосконалення останнього за рахунок часткової зміни нейромережевої архітектури та гіперпараметрів самої моделі.

Зокрема, було вирішено наступні поставлені задачі:

1. На основі аналізу підходів відображення текстових даних у векторну форму представлення було обґрунтовано вибір оптимальних за критерієм розмірності та ефективності паралельного обчислення.

2. На основі аналізу недоліків існуючих нейромережевих моделей та методів, що використовуються для задач автоматизованого перекладу природної мови сформульовано вимоги до побудови власної нейромережевої моделі.

3. Опрацьовано та сформовано навчальні та тестові вибірки текстових даних.

4. Обґрунтовано вибір програмного та апаратного забезпечення реалізації нейромережевої моделі у паралельних та розподілених системах.

5. Реалізовано запропоновану модель у вигляді комп’ютерної системи та здійснено верифікацію запропонованих підходів.

Підсумовуючи все вищесказане, варто зазначити, реалізована нейромережева модель здатна враховувати різноманітні граматичні конструкції мови, у тому числі звороти, а також стилістику тексту. Саме це надає змогу застосовувати нейронні мережі для вирішення задач вузькоспеціалізованого перекладу, наприклад, текстів конфесійного стилю, художнього, технічного напрямку, у тому числі медичної літератури.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Alvaro P. NMT-Keras: a Very Flexible Toolkit with a Focus on Interactive NMT and Online Learning [Електронний ресурс] / P. Álvaro, C. Francisco. – 2018. – Режим доступу до ресурсу: https://arxiv.org/pdf/1807.03096v3.pdf.

2. Attention Is All You Need [Електронний ресурс] / [V. Ashish, S. Noam, P. Niki та ін.]. – 2017. – Режим доступу до ресурсу: https://arxiv.org/pdf/1706.03762.pdf. - Дата доступу: 14.12.2020

3. BERT Deep Learning Model [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/google-research/bert. - Дата доступу: 14.12.2020

4. Changhan W. Neural Machine Translation with Byte-Level Subwords [Електронний ресурс] / W. Changhan, C. Kyunghyun, G. Jiatao. – 2019. – Режим доступу до ресурсу: https://arxiv.org/pdf/1909.03341.pdf. - Дата доступу: 14.12.2020

5. Character-based NMT with Transformer [Електронний ресурс] / G.Rohit,B. Laurent, D. Marc, G. Matthias // Arxiv. – 2019. – Режим доступу до ресурсу: https://arxiv.org/pdf/1911.04997.pdf. SubCharacter Chinese-English Neural Machine Translation with Wubi encoding [Електронний ресурс] / [Z. Wei, L. Feifei, W. Xiaodong та ін.] // Arxiv. – 2019. – Режим доступу до ресурсу: https://arxiv.org/pdf/1911.02737.pdf - Дата доступу: 14.12.2020

6. Delip R. Natural Language Processing with PyTorch: Build Intelligent Language Applications Using Deep Learning / R. Delip, M. Brian. – New York: O'Reilly Media, Inc, 2019. – 256 с.

7. Doc2Vec [Електронний ресурс] – Режим доступу до ресурсу:https://radimrehurek.com/gensim/models/doc2vec.html. - Дата доступу: 14.12.2020

8. Eli S. Deep Learning with PyTorch / S. Eli, A. Luca, V. Thomas. –Washington: Manning Publications, 2020. – 450 с.

9. Emily S. A Byte-sized Approach to Named Entity Recognition [Електронний ресурс] / S. Emily, N. Prem // Arxiv. – 2018. – Режим доступу до ресурсу: https://arxiv.org/pdf/1809.08386.pdf.

10. Flair NER Tagger [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/zalandoresearch/flair/blob/master/resources/docs/TUTORIAL_TAGGING.md. - Дата доступу: 14.12.2020

11. Lutskiv, A., Popovych, N. (2020) Big data approach to developing adaptable corpus tools CEUR Workshop Proceedings, 2604, pp. 374-395.

12. Lutskiv, A., Popovych, N. (2020) Big data-based approach to automated linguistic analysis effectiveness. In: Proceedings of the 2020 IEEE 3rd International Conference on Data Stream Mining and Processing, DSMP 2020, pp. 438-443

13. Lutskiv, A., Popovych, N.(2019) Adaptable text corpus development for specific linguistic research. In: 2019 IEEE International Scientific-Practical Conference: Problems of Infocommunications Science and Technology, PIC S and T 2019 - Proceedings, pp. 217-223.

14. MedLane: A Benchmark Dataset for Understandable Medical Language Translation [Електронний ресурс] / W.Yaqing, Y. Quanzeng, X. Cao, M. Fenglong. – 2020. – Режим доступу до ресурсу: https://arxiv.org/pdf/2012.02420.pdf. - Дата доступу: 14.12.2020

15. Natural Language Toolkit [Електронний ресурс] – Режим доступу до ресурсу: https://www.nltk.org/py-modindex.html. - Дата доступу: 14.12.2020

16. Popovych N., Lutskiv A., Tyshtchuk A. (2020) Corpus-Based Concept Translation in: Фаховий та художній переклад: теорія, методологія, практика: збірник наукових праць / за заг. ред. А.Г. Гудманяна, С.І. Сидоренка. К.: Аграр Медіа Груп,340 с. С. 306-314.

17. Stanford NER [Електронний ресурс] – Режим доступу до ресурсу: https://nlp.stanford.edu/software/. - Дата доступу: 14.12.2020

18. Universal Sentence Encoder [Електронний ресурс] – Режим доступу до ресурсу: https://tfhub.dev/google/universal-sentence-encoder/1. - Дата доступу: 14.12.2020

19. Word2Vec [Електронний ресурс] – Режим доступу до ресурсу: https://radimrehurek.com/gensim/tutorial.html. - Дата доступу: 14.12.2020

20. WordNet [Електронний ресурс] – Режим доступу до ресурсу: https://wordnet.princeton.edu/documentation. - Дата доступу: 14.12.2020

21. Марк Л. Python. Карманный справочник / Лутц Марк., 2016. – 320 с.

22. Массарон Л. Крупномасштабное машинное обучение вместе с Python / Л. Массарон, Б. Шарден, А. Боскетті., 2016. – 358 с.

23. Охеда Т. Прикладной анализ текстовых данных на Python / Т. Охеда, Б. Бенгфорт, Р. Білбро. – Санкт-Петербург: Питер, 2016. – 368 с.

24. Патрік Д. Искусственный интеллект с примерами на Python / Патрік., 2019. – 448 с.

25. Плас В. Python для сложных задач. Наука о данных и машинное обучение/ Вандер Плас. – Санкт-Петербург: Питер, 2016. – 576 с.

26. Шоллє Ф. Глубокое обучение на Python / Франуса Шоллє., 2016. – 400 с.

Додаток А

X Всеукраїнська студентська науково - технічна конференція
"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ТЕРНОПІЛЬСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ПУЛЮЯ
ТЕХНІЧНИЙ НАВЧАЛЬНО – НАУКОВИЙ ІНСТИТУТ
НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ БІОРЕСУРСІВ ТА
ПРИРОДОКОРИСТУВАННЯ
ЛЬВІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ІВАНА ФРАНКА
ЛУЦЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ
ЧЕРНІВЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ЮРІЯ ФЕДЬКОВИЧА
ДОНБАСЬКА ДЕРЖАВНА МАШИНОБУДІВНА АКАДЕМІЯ

Студентське наукове товариство

СНТ ТНТУ

X ВСЕУКРАЇНСЬКА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ.

АКТУАЛЬНІ ПИТАННЯ"

25-26 квітня 2017 р.

(збірник тез конференції)

ТОМ 1

Тернопіль 2017

УДК: 537.8 (07) (043)
Луцишин Р. ст. гр. СІ – 21
Тернопільський національний технічний університет імені Івана Пулюя

THE CHEAPEST AND MOST POWERFUL MICROPROCESSOR?

Scientific Supervisor: Perenchuk O.Z.

On the 2nd of March, 2017 hardware reviewers worldwide began posting their first reviews of Ryzen 7 1800X. Ryzen 7 CPUs were taken through virtually every conceivable CPU synthetic and real world productivity test or benchmarks, as well as a wide range of games at various resolutions and settings. Many independent reviewers have given their opinion about the release of the new microprocessor AMD Ryzen. So, let me introduce the varieties of this processor to you.

AMD Ryzen was released in 3 main types, they are : AMD Ryzen 3, AMD Ryzen 5 and AMD Ryzen 7.

AMD is trying to shake up the market with shockingly low prices for its 8C/16T Ryzen 7 line-up. And while these CPUs don't dominate every workload, there is hope the company's newest architecture is compelling across enough segments to put much-needed pressure on Intel. One component of AMD's strategy involves attractive pricing. The flagship Ryzen 7 1800X grabbed attention for its ability to battle Intel's Broadwell-E-based Core i7-6900K for $550 less (and with the same number of execution cores). We agree that the 1800X is compelling in threaded productivity and content creation apps. But we think you'll derive more value out from the cheaper Ryzen 7 1700X ($400) and 1700 ($330). The former goes up against Intel's $450 Core i7-6800K, while the latter undercuts Core i7-7700K. In both cases, AMD chips wield more processing resources than the Intel competition.

AMD claims that the gaming performance issues stem from how applications interact with the intricacies of its new architecture. The company expects a wave of updates from various developers that will eventually remedy this (though so far only two developments have publicly committed to optimizing their engines for the new processors). Until something concrete happens, though, we don't see much value in gaming-specific Ryzen 7 1800X builds. Might the Ryzen 7 1700X cast a more favorable light on gaming? After all, it costs $100 less, carries over the eight-core configuration with 16MB of L3 cache, and continues to offer an unlocked ratio multiplier.

The unlocked multiplier is especially interesting, given the similarities up and down the Ryzen 7 family. Given a similar 95W TDP between the $500 1800X and $400 1700X, then, the only technical differences between them are their base, two-core Precision Boost, and XFR clock rates. Out of the box, 1800X enjoys a 200 MHz advantage down low and up top. But we've heard claims that 1700X hits a similar ceiling as 1800X when it comes to overclocking.

Right out of the gate, Ryzen 7 1700X looks like a smarter buy than 1800X. But is it smart enough to maintain AMD's strong position in well-threaded desktop apps and make up some value ground in gaming, where the architecture isn't as strong.

Міністерство освіти і науки України,
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет імені
Івана Франка,
Гірничо-металургійна академія ім. Станіслава Сташиця
(Польща)
Луцький національний технічний університет,
Чернівецький національний університет
імені Юрія Федьковича,
Вроцлавський економічний університет (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ.

АКТУАЛЬНІ ПИТАННЯ"

26-27 квітня 2018 р.

(збірник тез конференції)

ТОМ 1

Тернопіль 2018

УДК: 537.8 (07) (043)
Луцишин Р. - ст. гр. СІ – 31
Тернопільський національний технічний університет імені Івана Пулюя

BENEFITS & RISKS OF ARTIFICIAL INTELLIGENCE

Supervisor: Perenchuk O.Z.

From SIRI to self-driving cars, artificial intelligence (AI) is progressing rapidly. While science fiction often portrays AI as robots with human-like characteristics, AI can encompass anything from Google's search algorithms to IBM's Watson to autonomous weapons.

Artificial intelligence today is properly known as narrow AI (or weak AI), in that it is designed to perform a narrow task (e.g. only facial recognition or only internet searches or only driving a car). However, the long-term goal of many researchers is to create general AI (AGI or strong AI). While narrow AI may outperform humans at whatever its specific task is, like playing chess or solving equations, AGI would outperform humans at nearly every cognitive task.

In the near term, the goal of keeping AI's impact on society beneficial motivates research in many areas, from economics and law to technical topics such as verification, validity, security and control. Whereas it may be little more than a minor nuisance if your laptop crashes or gets hacked, it becomes all the more important that an AI system does what you want it to do if it controls your car, your airplane, your pacemaker, your automated trading system or your power grid. Another short-term challenge is preventing a devastating arms race in lethal autonomous weapons.

In the long term, an important question is what will happen if the quest for strong AI succeeds and an AI system becomes better than humans at all cognitive tasks. As pointed out by I.J. Good in 1965, designing smarter AI systems is itself a cognitive task. Such a system could potentially undergo recursive self-improvement, triggering an intelligence explosion leaving human intellect far behind. By inventing revolutionary new technologies, such a superintelligence might help us eradicate war, disease, and poverty, and so the creation of strong AI might be the biggest event in human history. Some experts have expressed concern, though, that it might also be the last, unless we learn to align the goals of the AI with ours before it becomes superintelligent.

Most researchers agree that a superintelligent AI is unlikely to exhibit human emotions like love or hate, and that there is no reason to expect AI to become intentionally benevolent or malevolent. Instead, when considering how AI might become a risk, experts think two scenarios most likely:

The AI is programmed to do something devastating: Autonomous weapons are artificial intelligence systems that are programmed to kill. In the hands of the wrong person, these weapons could easily cause mass casualties. Moreover, an AI arms race could inadvertently lead to an AI war that also results in mass casualties. To avoid being thwarted by the enemy, these weapons would be designed to be extremely difficult to simply "turn off," so humans could plausibly lose control of such a situation. This risk is one that's present even with narrow AI, but grows as levels of AI intelligence and autonomy increase.

The AI is programmed to do something beneficial, but it develops a destructive method for achieving its goal: This can happen whenever we fail to fully align the AI's goals with ours, which is strikingly difficult. If you ask an obedient intelligent car to take you to the airport as fast as possible, it might get you there chased by helicopters and covered in vomit, doing not what you wanted but literally what you asked for. If a superintelligent system is tasked with a ambitious geoengineering project, it might wreak havoc with our ecosystem as a side effect, and view human attempts to stop it as a threat to be met.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Тернопільський національний технічний університет імені Івана Пулюя (Україна)
Національна академія наук України
Університет імені П'єра і Марії Кюрі (Франція)
Маріборский університет (Словения)
Технічний університет у Кошице (Словаччина)
Вільнюський технічний університет ім. Гедимінаса (Литва)
Шяуляйська державна колегія (Литва)
Жешувський політехнічний університет ім. Лукасевича (Польща)
Білоруський національний технічний університет (Республіка Білорусь)
Міжнародний університет цивільної авіації (Марокко)
Національний університет біоресурсів і природокористування України (Україна)
Наукове товариство ім. Шевченка
ГО «Асоціація випускників Тернопільського національного технічного університету імені Івана Пулюя»

АКТУАЛЬНІ ЗАДАЧІ СУЧАСНИХ ТЕХНОЛОГІЙ

Збірник

тез доповідей

Том ІІ

VII Міжнародної науково-технічної конференції молодих учених та студентів

28-29 листопада 2018 року

УКРАЇНА
ТЕРНОПІЛЬ – 2018

УДК: 004.272.3

А. М. Луцків канд. техн. наук, доц., Р. О. Луцишин
Тернопільський національний технічний університет імені Івана Пулюя, Україна

ВИКОРИСТАННЯ КОНТЕЙНЕРИЗАЦІЇ ПРИ РОЗРОБЛЕННІ ЛАБОРАТОРНИХ ПРАКТИКУМІВ СТУДЕНТІВ

A. M. Lutskiv Ph.D., Assoc. Prof., R.O. Lutsyshyn

CONTAINERIZATION USAGE FOR THE STUDENT'S LABORATORY WORKSHOPS DEVELOPMENT

При розробленні лабораторних практикумів перед викладачем постають наступні завдання:

- скорочення часу для підготовки та розгортання лабораторного стенду;
- спрощення процесу розгортання лабораторного стенду (студент може не мати хороших навичок системного адміністрування для встановлення та конфігурування певних типів сервісів, або вони розглядаються у інших курсах);
- мінімізація використання системних ресурсів;
- у ряді випадків доцільною є максимальна автоматизація та спрощення розгортання лабораторного стенду, зокрема, якщо мова йде про дистанційне навчання.

Віртуалізація на рівні операційної системи — метод віртуалізації, при якому ядро операційної системи підтримує декілька ізольованих примірників простору користувача, замість одного. Ці примірники (часто звані контейнерами або зонами) з точки зору користувача повністю ідентичні реальному серверові. Ядро забезпечує повну ізольованість контейнерів, тому програми з різних контейнерів не можуть впливати одна на одну.

При організації навчального процесу доволі часто можна зустріти підхід до розгортання лабораторних стендів у вигляді образів готових віртуальних машин (Hadoop-кластери Cloudera CDH та Hortonworks HDP, MongoDB University[1]). Проте, такі образи віртуальних машин є доволі об'ємними й їх використання є доволі ресурсоємним. У даному випадку використовується повна віртуалізація з використанням віртуальних машин Virtual Box, VM Ware.

На думку авторів, доцільнішим є використання технологій контейнеризації, зокрема, використання платформи Docker. Такий підхід передбачає наявність у студента деяких базових уявлень про контейнеризацію, проте, запуск та робота з таким лабораторним стендом є доволі простою (кількість необхідних попередніх налаштувань та кількість необхідних дій для запуску/зупинки/зберігання стану) та ефективною (з точки зору використання пам'яті й процесорного часу), швидшою у розгортанні (необхідність завантаження даних меншого об'єму), може бути використана як на робочому комп'ютері та і в хмарному сервісі.

Даний програмний продукт має наступні властивості:

- динамічна інфраструктура;
- швидке впровадження змін;
- спрощене розгортання додатків та програм.

Зазначимо, що існує кілька технологій контейнеризації на базі операційної системи (ОС) Linux, основними з яких є LXC (LinuX Container), а також LXD, CGManager, LXCFS. Для спрощення роботи з контейнерами використовують спеціалізовані програмні продукти, які в свою чергу базуються на контейнерах і представляють зручний інтерфейс взаємодії з ними, надають різноманітні засоби автоматизації роботи з ними. До складу інструментарію LXC входить бібліотека liblxc,

набір утиліт (lxc-create, lxc-start, lxc-stop, lxc-ls і тому подібне), шаблони для побудови контейнерів і набір біндінгів для різних мов програмування. Для ізоляції процесів, мережевого стека ipc, uts і точок монтування використовується механізм просторів імен (namespaces). Для обмеження ресурсів застосовуються cgroups. Для пониження привілеїв і обмеження доступу задіяні такі можливості ядра, як профілі Apparmor і SELinux, політики Seccomp, Chroots (pivot_root) і capabilities.

У контексті контейнерів Linux управління ресурсами організовано через cgroups . Cgroups дозволяють користувачеві виділяти ресурси, такі як процесорний час, системна пам'ять, пропускна здатність мережі, блок введення-виведення або будь-яку комбінацію з цих ресурсів для установки обмеженою користувачем групи завдань або процесів, запущених в даній системі. Як приклад, можна навести одну з популярних платформ для роботи з контейнерами — Docker. У цілому, використання даного підходу дає змогу створювати віртуальні одиниці більш ефективно і менш ресурсозатратно, у порівнянні зі створенням віртуальної машини.

Дане вирішення є апробованим при підготовці лабораторних практикумів з курсів «Паралельні та розподілені обчислення»[2], з використанням технологій OpenMP, MPI, OpenCL, CUDA та «Інформаційні системи паралельної та розподіленої обробки даних» (використання веб-сервісів, систем опрацювання великих даних).

Використання даного підходу дає змогу викладачеві використовувати офіційні репозиторії компаній Nvidia, AMD, Intel та інших для отримання вже налаштованих та відлагоджених контейнерів з наборами системного програмного забезпечення для задач паралельного програмування та попередньоналаштованими хмарними сервісами. Використовуючи базові механізми багатошаровості Docker-контейнерів, викладач, може відносно просто інтегрувати в контейнер лабораторне завдання й необхідні елементи його виконання. Публікація в публічному репозиторії дає змогу студенту, відносно просто, завантажити контейнери та працювати з ними. При виконанні таких лабораторних практикумів, студенти здобувають додатково необхідні як розробникам так і системним адміністраторам навички DevOps-інженерів.

Варто відзначити, що операційна система Windows 10, також дає змогу запускати Linux-контейнери, що суттєво знижує поріг входження у предметну область для більшості студентів, які, на жаль, не мають досвіду роботи з Unix-подібними операційними системами.

Літератра

1. Turnbull J. The docker book containerization is the new virtualization / James Turnbull., 2014. – 321 c.

2. MongoDB University. M202: MongoDB Advanced Deployment and Operations. [Електронний ресурс] – Режим доступу до ресурсу: URL: https://university.mongodb.com/courses/M202/about

3. Docker Documents [Електронний ресурс] – Режим доступу до ресурсу: URL: https://docs.docker.com/.

4. Луцків А.М. Паралельні та розподілені обчислення : Навчальний посібник / Луцків А.М., Лупенко С.А., Пасічник В.В. — Львів : Магнолія 2006 , 2015 — 566 с. — ISBN 9786175741108.

Міністерство освіти і науки України
Національний педагогічний університет імені М.П. Драгоманова
Інститут інформаційних технологій і засобів навчання
Національної академії педагогічних наук
Рівненський ІТ-Кластер
Рівненський державний гуманітарний університет

МАТЕРІАЛИ
XIII Всеукраїнської науково-практичної конференції
«ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В ПРОФЕСІЙНІЙ ДІЯЛЬНОСТІ»

18 листопада 2020 року
м. Рівне

ЧАСТИНА 2.
ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ В СУСПІЛЬНО-ГУМАНІТАРНИХ НАУКАХ

ЧАСТИНА 3.
ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ
В ПРИРОДНИЧО-МАТЕМАТИЧНИХ ТА ЕКОНОМІЧНИХ НАУКАХ

ОЦІНКА СЕМАНТИЧНОЇ СХОЖОСТІ ТЕКСТОВИХ ДАНИХ НА ОСНОВІ МОДИФІКОВАНОЇ МЕТРИКИ ВІДСТАНІ СЛІВ У ЇЄРАРХІЇ СИНОНІМІЧНОГО ДЕРЕВА ПАКЕТУ NLTK

Сіткар Тарас Вікторович,
кандидат педагогічних наук, викладач кафедри комп'ютерних технологій
Тернопільський національний педагогічний університет імені Володимира Гнатюка
Луцишин Роман,
магістрант факультету комп'ютерних технологій
Тернопільський національний технічний університет імені Івана Пулюя

Анотація. *Дане дослідження присвячене опису та створенні алгоритму для оцінки семантичної схожості текстових даних на основі модифікованої метрики відстані слів у їєрархії синонімічного дерева пакету NLTK програмного середовища Python. Зокрема, проаналізовано можливості бібліотеки NLTK та семантичного словника для англійської мови WordNet програмного середовища Python. У статті було подано опис алгоритму для оцінки семантичної схожості текстових даних на основі модифікованої метрики відстані слів у їєрархії синонімічного дерева пакету NLTK, а також подано його реалізацію за допомогою програмного середовища Python.*
Ключові слова: *NLTK, WordNet, Wup similarity, semantic similarity of textual data, Python.*

Sitkar T., Lutsyshyn R. THE ESTIMATION OF SEMANTIC SIMILARITY OF TEXT DATA BASED ON MODIFIED WORDS DISTANCE METRIC IN THE SYNONYMOUS TREE HIERARCHY OF NLTK PACKAGE

Abstract. *This study deals with describing and creating an algorithm for evaluating of emantic similarity of text data based on modified words distance metric in the synonymous tree hierarchy of NLTK package of Python software. In particular, the features of NLTK library and the semantic dictionary for English WordNet of Python software have been analyzed. Algorithm description for evaluating of semantic similarity of text data based on modified words distance metric in the synonymous tree hierarchy of NLTK package of software Python has been presented. Its implementation via Python software has been submitted.*
Keywords. *NLTK, WordNet, Wup similarity, semantic similarity of text data, Python.*

Всі інтелектуальні системи обробки тексту природною мовою, на сучасному етапі, знайшли своє застосування в різних сферах і вирішують конкретні завдання. До систем, що використовуються автоматичну обробку тексту можна віднести: видавничі системи [1]; автоматизовані лексикографічні системи підготовки та використання словників [2]; інформаційно-пошукові системи [3]; системи машинного перекладу (МП) [4, 5]; системи розуміння і розпізнавання мови [6] тощо.

Більшість досліджень автоматичної обробки природної мови спрямовані на додатки і системи, які можуть бути згруповані на системи автоматичної категоризації текстів [7], системи антиплагіату і системи реферування серед інші; в яких текстове порівняння, в залежності від обставин, розглядається в якості другорядного завдання. Крім того, коли проводяться дослідження саме по темі текстового порівняння, воно проводиться для цілей систем інформаційного пошуку і антиплагіату. У зв'язку з цим не проводиться глибоких досліджень семантичного змісту оброблюваних текстів.

В даний час пошук подібності між текстами має велике практичне використання, в тому числі для виявлення плагіату в наукових і творчих дослідженнях. В роботі [8] згадуються три основні категорії виявлення текстуального подібності: порівняння на основі слів, лінійний пошук на підставі пунктів, що використовується пошуковими системами і стилістичний аналіз. Також існують методи, засновані на різних характеристиках текстів, такі як методи, засновані на семантиці, як для виявлення плагіату [9-11]; так і для пошуку інформації, так як це відображено в роботі [12].

На вході процесу порівняння текстів є два документи, призначені для порівняння, один з яких є еталоном. На першому рівні аналізу проводиться витяг текстових пасажів, як це описано в роботі [13], виходом цього першого рівня буде перелік значущих пасажів з кожного документа, які послужать як вхідних даних для наступного рівня для дозволу анафори [14] а останній, в свою чергу, надходить на рівень семантичного уявлення схем [12]. Побудована схема уявлення є входом для виявлення рівня семантичного подібності між текстовими пасажами.

На цьому рівні, в якості вхідних даних використовуються текстові пасажі з документів і на основі їх результатів порівняння, виконується обчислення подібності між цими двома документами.

Основним завдання дослідження полягало в описі та створенні алгоритмів для оцінки семантичної схожості текстових даних на основі модифікованої метрики відстані слів у ієрархії синонімічного дерева пакету NLTK.

Бібліотека NLTK – пакет бібліотек і програм для символьної і статистичної обробки природної мови, написаний на мові програмування Python.

Для пошуку семантичної схожості між двома словами за допомогою пакету інструментів NLTK використовуються так звані множини синонімів (synset – synonym set). Дані множини містять у собі набір слів, схожих за значенням до заданого, причому зберігаючи інформацію про частину мови та деякі інші властивості останнього. Усі групи слів, які знаходяться у бібліотеці NLTK розташовані у суворо визначеній ієрархії, подібній дереву, де кількість переходів по цьому дереву від одного слова до іншого визначає семантичну близькість між ними.

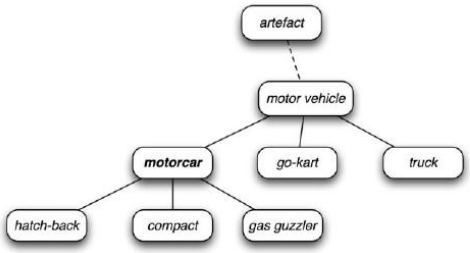

Рис. 1. Вигляд ієрархічного дерева термінів у мережі NLTK

WordNet – семантичний словник для англійської мови. У ньому слова англійської мови розбито на групи синонімів – синсети (від англ. synset, synonym set), та надається коротке загальне визначення, та семантичні відношення між цими словами.

Основою створення нової метрики став пошук семантичної близькості між кожним словом у першому тексті з кожним словом у другому тексті, після чого слід вибрати середнє значення.

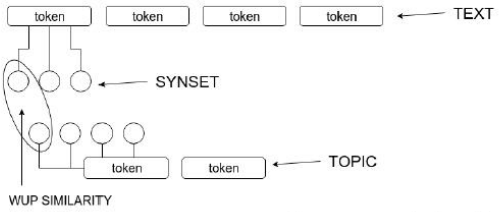

Рис. 2. Концепція обчислення семантичної схожості між двома реченнями

Wup similarity – це метрика, заснована Wu Palmer, яка базується на обчисленні спорідненості, враховуючи глибину двох синсетів у таксономіях WordNet, а також глибину LCS (Найменший загальний підрядник) (1):

$$\text{Wu} - \text{Palmer} = \frac{\text{depth}(\text{lcs}(s1, s2))}{\text{depth}(s1) + \text{depth}(s2)} \quad (1)$$

Приклад практичної реалізації даної метрики наведено нижче, у коді на високорівневій мові прогрумування Python.

Для обчислення схожості двох синсетів була створена наступна функція (рис. 3):

```
def calc_words_sim(self,word_1, word_2):
    '''
    This func used to find wupsimilarity for two words if synstes are
available return max value of similarity
    if synsets are not in available use edit distance
    '''
    word_1_synsets = self.get_synset(word_1)
    word_2_synsets = self.get_synset(word_2)
    if len(word_1_synsets) == 0 or len(word_2_synsets) == 0:
    return self.calc_edit_score(word_1, word_2)
    else:
    sim_all_synsets = []
    for word_1_synset in word_1_synsets:
    for word_2_synset in word_2_synsets:
    if word_1_synset.pos() == word_2_synset.pos(): #Use POS to speed up
the process
    synset_sim = self.calc_sim_synsets(word_1_synset,word_2_synset)
    if synset_sim:
    sim_all_synsets.append(synset_sim)
    else:
    sim_all_synsets.append(0)
    else:
    sim_all_synsets.append(0)
```

Рис. 3. Обчислення схожості двох синсетів

Для обчислення семантичної схожості набору синсетів (тексту) до іншого набору синсетів ми створили відповідну функцію за допомогою мови програмування Pyhton. Дана функція представлена на рис. 4:

```
def calc_phrase_sim(self,text, topic):
    '''
    :param text -- can be sentence or some sentences
    :param topic -- can be as a word and as a some words
    '''
    if len(text) == 0 or len(topic) == 0:
    return 0
    text_to_words = self.preprocess(text)
    text_to_words = word_tokenize(text) #Preprocess the text
    text_to_words = self.remove_stop_words(text_to_words)

    topic_to_words = self.preprocess(topic)
    topic_to_words = word_tokenize(topic) #Preprocess the topic if it has
the len more than 1 word
    topic_to_words = self.remove_stop_words(topic_to_words)
    #Calculate topic similarity to text
    topic_all_sim = [self.trg_word_sim_score(text_to_words,topic_) for
topic_ in topic_to_words]
    if len(topic_all_sim) > 0:
    topic_all_sim = np.mean(topic_all_sim)
    else:
    topic_all_sim = 0
    #Calculate text similarity to topic
    text_all_sim = [self.trg_word_sim_score(topic_to_words, text_) for
text_ in text_to_words]
    if len(text_all_sim) > 0:
    text_all_sim = np.mean(text_all_sim)
    else:
    text_all_sim = 0
    return np.mean([topic_all_sim, text_all_sim])
```

Рис. 4. Обчислення семантичної схожості набору синсетів(тексту) до іншого набору синсетів

Завершальним етапом є представлення результату та оцінка отриманої метрики, нижче наведений код перетворення стандартних одиниць побідності NLTK у свою метрику, у даному випадку відстань між текстами:

```
def convert_sim_to_dist(self,phrases_sim):¶
    '''¶
    Func to convert similarity into distance between text and topic¶
    '''¶
    return 20 if phrases_sim == 0 else abs(math.log(phrases_sim,0.8))¶
```

Рис. 5. Представлення результату та оцінка отриманої метрики

Таким чином, чим більше тексти є семантично схожими, тим менша відстань між ними буде отримана в результаті її обчислення та навпаки, чим більша відстань між текстами – тим більше вони семантично несхожі між собою.

Список використаних джерел

1. Yazykoznanie. Bol. entsikl. slovar' [Linguistics. Big encyclopaedic dictionary], chief ed. V. N. Yartseva. 2nd ed. Moscow: Bol. ros. entsikl., 1998, 685 p.

2. Marchuk Yu.N. Komp'yuternaya lingvistika [Computational linguistics]. Moscow: AST; Vostok-Zapad, 2007, 317 p.

3. Gaydamakin N.A. Avtomatizirovannye informatsionnye sistemy, bazy i banki dannykh. Vvodnyy kurs: ucheb. posobie [Automated information systems, databases and data. Introductory course: textbook]. Moscow: Gelios ARV, 2002, 368 p.

4. Baranov A.N. Vvedenie v prikladnuyu lingvistiku: ucheb. posobie [Introduction to applied linguistics: textbook]. Moscow: Editorial URSS, 2001, 360 p.

5. Iskusstvennyy intellect [Artificial intelligence]. In 3 book. Book 1. Sistemy obshcheniya I ekspertnye sistemy: spravochnik [Communication and expert systems: a Handbook], ed. By E.V. Popova. Moscow: Radio i svyaz', 1990, 464 p.

6. Potapova R.K. Rech': kommunikatsiya, informatsiya, kibernetika: ucheb. posobie [Speech: communication, information, cybernetics: textbook]. Moscow: Editorial URSS, 2003, 568 p.

7. Muñoz T.R. Representación del conocimiento textual mediante técnicas lógico-conceptuales en aplicaciones de tecnologías del lenguaje humano, Tesis doctoral. Universidad de Alicante. España, 2009, 128 p.

8. Maurer H., Kappe F. y Zaka B. Plagiarism – A Survey, Journal of Universal Computer Science, 2006, No. 12, pp. 1050-1084.

9. Bao J-P., Shen J-Y., Liu X-D., Liu H-Y. y Zhang X-D. Semantic Sequence Kin: A Method of Document Copy Detection, Advances In Knowledge Discovery and Data Mining. Lecture Notes in Artificial Intelligence (LNAI). Sydney, Australia, 2004, Vol. 3056, pp. 529-538.

10. Bao J-P., Shen J-Y., Liu X-D., Liu H-Y. y Zhang X-D. Finding Plagiarism Based on Common Semantic Sequence Model, The 5th International Conference on Advances in Web-Age Information Management (WAIM). Lecture Notes in Computer Science. China: Dalian, 2004, Vol. 3129, pp. 640-645.

11. Chi-Hong L. y Yuen-Yan C. A Natural Language Processing Approach to Automatic Plagiarism Detection, The 8th ACM Conference on Information Technology Education (SIGITE'07) – Florida, USA, 2007, pp. 213-218.

12. Vishnyakov R. Yu. Razrabotka i issledovanie formalizovannykh predstavleniy i semanticheskikh skhem predlozheniy tekstov nauchno-tekhnicheskogo stilya dlya povysheniya effektivnosti informatsionnogo poiska: diss. … kand. tekhn. nauk [The development and study of formal representations and semantic diagrams of the sentences of the texts of scientifictechnical style to improve the efficiency of information retrieval. Cand. of eng. sc. diss.]. Taganrog, 2012.

13. Bermudes S.Kh.G. O metode izvlecheniya znachimykh tekstovykh passazhey kak bazy dlya tekstovogo sravneniya [On the method of extraction of important text passages as a basis for tech-stowage comparison], Informatizatsiya i svyaz' [Informatization and communication], 2016, No. 3, pp. 231-219.

14. Salguero L. F. Resolución abductiva de anáforas pronominales. Available at: http://www.http://personal.us.es/fsoler/papers/ivjornadas.pdf. (accessed 29 January 2016).

TITA 2020

Translation Conference

Kyiv, 11-12 December 2020

Certificate of participation

awarded to

Roman Lutsyshyn

International Research-to-Practice Conference for Translators, Young Scholars and Students "Translation Industry: Theory in Action"

Institute of Philology,
Taras Shevchenko National University of Kyiv
Kyiv, 11-12 December 2020

Dr. habil., Full Prof. Hryhorii Semeniuk
Head of Organising Committee
Director of the Institute of Philology
Taras Shevchenko National University of Kyiv

Nataliya Popovych, PhD
Andriy Lutskiv, PhD
Roman Lutsyshyn
State University "Uzhhorod National University"
Ternopil Ivan Puluj National Technical University

CORPUS-BASED TRANSLATION FOR RESOLVING SPECIFIC LINGUISTIC TASKS: TYPES OF CORPORA VS TYPES OF TRANSLATION ISSUES

There are many common tasks in translation studies which arise today. Preparation of draft translation, translation, proofreading are just a few to name. Nowadays all these tasks could be resolved by means of semi or fully automated CAT tools.The specific fields of such issues can vary, i.e., computational linguistics, machine learning, deep learning and other supporting information technologies. In this case the role of translator should be shifted from translator to *proofreader and expert in translation quality assessment*.

To meet a specific modern CAT tool main requirements the following tasks should be resolved:

1. choosing and preparation of the best texts for training process;
2. text labeling (automate or semiautomate);
3. text feature extraction;
4. text encoding (binarization);
5. training neural network and choosing proper hyperparameters of it;
6. applying this trained neural network in translation tool.

These findings cover the process of building custom tools for text translation in some domains with a higher level of translation accuracy. This tool can comprise:

- Corpus tool for building custom parallel corpora for feature extraction. This corpus tool could be implemented with Big Data approaches (Lutskiv, Popovych, 2019, 2020)
- Set of tools for neural networks for supervised learning for building trained neural networks. Also this tool estimates qualitative characteristics of translation accuracy and equivalence on different levels.
- Translation tool which comprises the trained neural network, corpora as a supplement tool and proofreading tool. This tool allows translators to load and translate texts with different supplement features provided by corpora tool.

Suggested approach can reduce the amount of the translator's routine work by full draft translation of the document in the case of the simple text translating and prepare draft translation for proofreading in the case of complex text translating issues.

References:

Popovych N., Lutskiv A., Tyshtchuk A. (2020) *Corpus-Based Concept Translation* in: Фаховий та художній переклад: теорія, методологія, практика: збірник наукових праць / за заг. ред. А.Г. Гудманяна, С.І. Сидоренка. К.: Аграр Медіа Груп,340 с. С. 306-314.

Lutskiv, A., Popovych, N. (2020) *Big data-based approach to automated linguistic analysis effectiveness.* In: Proceedings of the 2020 IEEE 3rd International Conference on Data Stream Mining and Processing, DSMP 2020, pp. 438-443

Lutskiv, A., Popovych, N. (2020) *Big data approach to developing adaptable corpus tools* CEUR Workshop Proceedings, 2604, pp. 374-395.

Lutskiv, A., Popovych, N.(2019) *Adaptable text corpus development for specific linguistic research.* In: 2019 IEEE International Scientific-Practical Conference: Problems of Infocommunications Science and Technology, PIC S and T 2019 - Proceedings, pp. 217-223

Додаток Б

Лістинг коду програми

```
import math
import os
import warnings
from dataclasses import dataclass
from typing import Optional, Tuple

import torch
import torch.utils.checkpoint
from torch import nn
from torch.nn import CrossEntropyLoss, MSELoss

from ...activations import ACT2FN
from ...file_utils import (
    ModelOutput,
    add_code_sample_docstrings,
    add_start_docstrings,
    add_start_docstrings_to_model_forward,
    replace_return_docstrings,
)
from ...modeling_outputs import (
    BaseModelOutputWithCrossAttentions,
    BaseModelOutputWithPoolingAndCrossAttentions,
    CausalLMOutputWithCrossAttentions,
    MaskedLMOutput,
    MultipleChoiceModelOutput,
    NextSentencePredictorOutput,
    QuestionAnsweringModelOutput,
    SequenceClassifierOutput,
    TokenClassifierOutput,
)
from ...modeling_utils import (
    PreTrainedModel,
    apply_chunking_to_forward,
    find_pruneable_heads_and_indices,
    prune_linear_layer,

)
from ...utils import logging
from .configuration_bert import BertConfig


logger = logging.get_logger(__name__)

_CONFIG_FOR_DOC = "BertConfig"
_TOKENIZER_FOR_DOC = "BertTokenizer"

BERT_PRETRAINED_MODEL_ARCHIVE_LIST = [
    "bert-base-uncased",
```

```
    "bert-large-uncased",
    "bert-base-cased",
    "bert-large-cased",
    "bert-base-multilingual-uncased",
    "bert-base-multilingual-cased",
    "bert-base-chinese",
    "bert-base-german-cased",
    "bert-large-uncased-whole-word-masking",
    "bert-large-cased-whole-word-masking",
    "bert-large-uncased-whole-word-masking-finetuned-squad",
    "bert-large-cased-whole-word-masking-finetuned-squad",
    "bert-base-cased-finetuned-mrpc",
    "bert-base-german-dbmdz-cased",
    "bert-base-german-dbmdz-uncased",
    "cl-tohoku/bert-base-japanese",
    "cl-tohoku/bert-base-japanese-whole-word-masking",
    "cl-tohoku/bert-base-japanese-char",
    "cl-tohoku/bert-base-japanese-char-whole-word-masking",
    "TurkuNLP/bert-base-finnish-cased-v1",
    "TurkuNLP/bert-base-finnish-uncased-v1",
    "wietsedv/bert-base-dutch-cased",
    # See all BERT models at https://huggingface.co/models?filter=bert
]


def load_tf_weights_in_bert(model, config, tf_checkpoint_path):

"""Load tf checkpoints in a pytorch model."""
    try:
        import re

        import numpy as np
        import tensorflow as tf
    except ImportError:
        logger.error(
            "Loading a TensorFlow model in PyTorch, requires
TensorFlow to be installed. Please see "
            "https://www.tensorflow.org/install/ for installation
instructions."
        )
        raise
    tf_path = os.path.abspath(tf_checkpoint_path)
    logger.info("Converting TensorFlow checkpoint from
{}".format(tf_path))
    # Load weights from TF model
    init_vars = tf.train.list_variables(tf_path)
    names = []
    arrays = []
    for name, shape in init_vars:
        logger.info("Loading TF weight {} with shape {}".format(name,
shape))
        array = tf.train.load_variable(tf_path, name)
        names.append(name)
```

```
        arrays.append(array)

    for name, array in zip(names, arrays):
        name = name.split("/")
        #    adam_v    and    adam_m    are    variables    used    in
AdamWeightDecayOptimizer to calculated m and v
        # which are not required for using pretrained model
        if any(
            n  in  ["adam_v",  "adam_m",  "AdamWeightDecayOptimizer",
"AdamWeightDecayOptimizer_1", "global_step"]
            for n in name
        ):
            logger.info("Skipping {}".format("/".join(name)))
            continue

pointer = model
        for m_name in name:
            if re.fullmatch(r"[A-Za-z]+_\d+", m_name):
                scope_names = re.split(r"_(\d+)", m_name)
            else:
                scope_names = [m_name]
            if scope_names[0] == "kernel" or scope_names[0] == "gamma":
                pointer = getattr(pointer, "weight")
            elif scope_names[0] == "output_bias" or scope_names[0] ==
"beta":
                pointer = getattr(pointer, "bias")
            elif scope_names[0] == "output_weights":
                pointer = getattr(pointer, "weight")
            elif scope_names[0] == "squad":
                pointer = getattr(pointer, "classifier")
            else:
                try:
                    pointer = getattr(pointer, scope_names[0])
                except AttributeError:
                    logger.info("Skipping {}".format("/".join(name)))
                    continue
            if len(scope_names) >= 2:
                num = int(scope_names[1])
                pointer = pointer[num]
        if m_name[-11:] == "_embeddings":
            pointer = getattr(pointer, "weight")
        elif m_name == "kernel":
            array = np.transpose(array)
        try:
            assert (
                pointer.shape == array.shape
            ),  f"Pointer  shape  {pointer.shape}  and  array  shape
{array.shape} mismatched"
        except AssertionError as e:
            e.args += (pointer.shape, array.shape)
            raise
        logger.info("Initialize PyTorch weight {}".format(name))
        pointer.data = torch.from_numpy(array)
```

```
    return model


class BertEmbeddings(nn.Module):
    """Construct the embeddings from word, position and token_type
embeddings."""

    def __init__(self, config):
        super().__init__()
        self.word_embeddings     =     nn.Embedding(config.vocab_size,
config.hidden_size, padding_idx=config.pad_token_id)
        self.position_embeddings                                        =
nn.Embedding(config.max_position_embeddings, config.hidden_size)
        self.token_type_embeddings                                      =
nn.Embedding(config.type_vocab_size, config.hidden_size)

        # self.LayerNorm is not snake-cased to stick with TensorFlow
model variable name and be able to load
        # any TensorFlow checkpoint file
        self.LayerNorm       =       nn.LayerNorm(config.hidden_size,
eps=config.layer_norm_eps)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)

        # position_ids (1, len position emb) is contiguous in memory
and exported when serialized
        self.register_buffer("position_ids",
torch.arange(config.max_position_embeddings).expand((1, -1)))
        self.position_embedding_type         =         getattr(config,
"position_embedding_type", "absolute")

    def     forward(self,     input_ids=None,     token_type_ids=None,
position_ids=None, inputs_embeds=None):
        if input_ids is not None:
            input_shape = input_ids.size()
        else:
            input_shape = inputs_embeds.size()[:-1]

        seq_length = input_shape[1]

        if position_ids is None:
            position_ids = self.position_ids[:, :seq_length]

        if token_type_ids is None:
            token_type_ids = torch.zeros(input_shape, dtype=torch.long,
device=self.position_ids.device)

        if inputs_embeds is None:
            inputs_embeds = self.word_embeddings(input_ids)
        token_type_embeddings                                           =
self.token_type_embeddings(token_type_ids)

        embeddings = inputs_embeds + token_type_embeddings
        if self.position_embedding_type == "absolute":
```

```
            position_embeddings                                         =
self.position_embeddings(position_ids)
            embeddings += position_embeddings
        embeddings = self.LayerNorm(embeddings)
        embeddings = self.dropout(embeddings)
        return embeddings


class BertSelfAttention(nn.Module):
    def __init__(self, config):
        super().__init__()
        if  config.hidden_size  %  config.num_attention_heads  !=  0  and
not hasattr(config, "embedding_size"):
            raise ValueError(
                "The hidden size (%d) is not a multiple of the number
of attention "
                "heads       (%d)"       %       (config.hidden_size,
config.num_attention_heads)
            )

        self.num_attention_heads = config.num_attention_heads
        self.attention_head_size     =     int(config.hidden_size     /
config.num_attention_heads)
        self.all_head_size      =      self.num_attention_heads      *
self.attention_head_size

        self.query = nn.Linear(config.hidden_size, self.all_head_size)
        self.key = nn.Linear(config.hidden_size, self.all_head_size)

        self.value = nn.Linear(config.hidden_size, self.all_head_size)

        self.dropout = nn.Dropout(config.attention_probs_dropout_prob)
        self.position_embedding_type         =         getattr(config,
"position_embedding_type", "absolute")
        if    self.position_embedding_type    ==    "relative_key"    or
self.position_embedding_type == "relative_key_query":
            self.max_position_embeddings                                =
config.max_position_embeddings
            self.distance_embedding      =      nn.Embedding(2      *
config.max_position_embeddings - 1, self.attention_head_size)

    def transpose_for_scores(self, x):
        new_x_shape   =   x.size()[:-1]   +   (self.num_attention_heads,
self.attention_head_size)
        x = x.view(*new_x_shape)
        return x.permute(0, 2, 1, 3)

    def forward(
        self,
        hidden_states,
        attention_mask=None,
        head_mask=None,
        encoder_hidden_states=None,
```

```
        encoder_attention_mask=None,
        output_attentions=False,
    ):
        mixed_query_layer = self.query(hidden_states)

        # If this is instantiated as a cross-attention module, the
keys
        # and values come from an encoder; the attention mask needs to
be
        # such that the encoder's padding tokens are not attended to.
        if encoder_hidden_states is not None:
            mixed_key_layer = self.key(encoder_hidden_states)
            mixed_value_layer = self.value(encoder_hidden_states)
            attention_mask = encoder_attention_mask
        else:
            mixed_key_layer = self.key(hidden_states)
            mixed_value_layer = self.value(hidden_states)

        query_layer = self.transpose_for_scores(mixed_query_layer)
        key_layer = self.transpose_for_scores(mixed_key_layer)
        value_layer = self.transpose_for_scores(mixed_value_layer)

        # Take the dot product between "query" and "key" to get the
raw attention scores.
        attention_scores          =          torch.matmul(query_layer,
key_layer.transpose(-1, -2))

        if    self.position_embedding_type    ==    "relative_key"    or
self.position_embedding_type == "relative_key_query":
            seq_length = hidden_states.size()[1]
            position_ids_l = torch.arange(seq_length, dtype=torch.long,
device=hidden_states.device).view(-1, 1)
            position_ids_r = torch.arange(seq_length, dtype=torch.long,
device=hidden_states.device).view(1, -1)
            distance = position_ids_l - position_ids_r
            positional_embedding  =  self.distance_embedding(distance  +
self.max_position_embeddings - 1)
            positional_embedding                                        =
positional_embedding.to(dtype=query_layer.dtype)  # fp16 compatibility

            if self.position_embedding_type == "relative_key":
                relative_position_scores    =    torch.einsum("bhld,lrd-
>bhlr", query_layer, positional_embedding)
                attention_scores       =       attention_scores       +
relative_position_scores
            elif self.position_embedding_type == "relative_key_query":
                relative_position_scores_query                          =
torch.einsum("bhld,lrd->bhlr", query_layer, positional_embedding)
                relative_position_scores_key  =  torch.einsum("bhrd,lrd-
>bhlr", key_layer, positional_embedding)
                attention_scores       =       attention_scores       +
relative_position_scores_query + relative_position_scores_key
```

```
        attention_scores = attention_scores / math.sqrt(self.attention_head_size)
        if attention_mask is not None:

    # Apply the attention mask is (precomputed for all layers in BertModel forward() function)
            attention_scores = attention_scores + attention_mask

        # Normalize the attention scores to probabilities.
        attention_probs = nn.Softmax(dim=-1)(attention_scores)

        # This is actually dropping out entire tokens to attend to, which might
        # seem a bit unusual, but is taken from the original Transformer paper.
        attention_probs = self.dropout(attention_probs)

        # Mask heads if we want to
        if head_mask is not None:
            attention_probs = attention_probs * head_mask

        context_layer = torch.matmul(attention_probs, value_layer)

        context_layer = context_layer.permute(0, 2, 1, 3).contiguous()
        new_context_layer_shape = context_layer.size()[:-2] + (self.all_head_size,)
        context_layer = context_layer.view(*new_context_layer_shape)

        outputs = (context_layer, attention_probs) if output_attentions else (context_layer,)
        return outputs


class BertSelfOutput(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        self.LayerNorm = nn.LayerNorm(config.hidden_size, eps=config.layer_norm_eps)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)

    def forward(self, hidden_states, input_tensor):

    hidden_states = self.dense(hidden_states)
        hidden_states = self.dropout(hidden_states)
        hidden_states = self.LayerNorm(hidden_states + input_tensor)
        return hidden_states


class BertAttention(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.self = BertSelfAttention(config)
```

```
        self.output = BertSelfOutput(config)
        self.pruned_heads = set()

    def prune_heads(self, heads):
        if len(heads) == 0:
            return
        heads, index = find_pruneable_heads_and_indices(
            heads,                          self.self.num_attention_heads,
self.self.attention_head_size, self.pruned_heads
        )

        # Prune linear layers
        self.self.query = prune_linear_layer(self.self.query, index)
        self.self.key = prune_linear_layer(self.self.key, index)
        self.self.value = prune_linear_layer(self.self.value, index)
        self.output.dense    =    prune_linear_layer(self.output.dense,
index, dim=1)

        # Update hyper params and store pruned heads
        self.self.num_attention_heads  =  self.self.num_attention_heads
- len(heads)
        self.self.all_head_size   =   self.self.attention_head_size   *
self.self.num_attention_heads
        self.pruned_heads = self.pruned_heads.union(heads)

    def forward(
        self,
        hidden_states,

      attention_mask=None,
        head_mask=None,
        encoder_hidden_states=None,
        encoder_attention_mask=None,
        output_attentions=False,
    ):
        self_outputs = self.self(
            hidden_states,
            attention_mask,
            head_mask,
            encoder_hidden_states,
            encoder_attention_mask,
            output_attentions,
        )
        attention_output = self.output(self_outputs[0], hidden_states)
        outputs  =  (attention_output,)  +  self_outputs[1:]    #  add
attentions if we output them
        return outputs


class BertIntermediate(nn.Module):
    def __init__(self, config):
        super().__init__()
```

```
        self.dense           =           nn.Linear(config.hidden_size,
config.intermediate_size)
        if isinstance(config.hidden_act, str):
            self.intermediate_act_fn = ACT2FN[config.hidden_act]
        else:
            self.intermediate_act_fn = config.hidden_act

    def forward(self, hidden_states):
        hidden_states = self.dense(hidden_states)
        hidden_states = self.intermediate_act_fn(hidden_states)
        return hidden_states


class BertOutput(nn.Module):
    def __init__(self, config):
        super().__init__()

      self.dense         =         nn.Linear(config.intermediate_size,
config.hidden_size)
        self.LayerNorm       =       nn.LayerNorm(config.hidden_size,
eps=config.layer_norm_eps)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)

    def forward(self, hidden_states, input_tensor):
        hidden_states = self.dense(hidden_states)
        hidden_states = self.dropout(hidden_states)
        hidden_states = self.LayerNorm(hidden_states + input_tensor)
        return hidden_states


class BertLayer(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.chunk_size_feed_forward = config.chunk_size_feed_forward
        self.seq_len_dim = 1
        self.attention = BertAttention(config)
        self.is_decoder = config.is_decoder
        self.add_cross_attention = config.add_cross_attention
        if self.add_cross_attention:
            assert  self.is_decoder,  f"{self}  should  be  used  as  a
decoder model if cross attention is added"
            self.crossattention = BertAttention(config)
        self.intermediate = BertIntermediate(config)
        self.output = BertOutput(config)

    def forward(
        self,
        hidden_states,
        attention_mask=None,
        head_mask=None,
        encoder_hidden_states=None,
        encoder_attention_mask=None,
        output_attentions=False,
```

```
    ):
        self_attention_outputs = self.attention(
            hidden_states,

           attention_mask,
            head_mask,
            output_attentions=output_attentions,
        )
        attention_output = self_attention_outputs[0]
        outputs = self_attention_outputs[1:]  # add self attentions if
we output attention weights

        if self.is_decoder and encoder_hidden_states is not None:
            assert hasattr(
                self, "crossattention"
            ), f"If `encoder_hidden_states` are passed, {self} has to
be    instantiated    with    cross-attention    layers    by    setting
`config.add_cross_attention=True`"
            cross_attention_outputs = self.crossattention(
                attention_output,
                attention_mask,
                head_mask,
                encoder_hidden_states,
                encoder_attention_mask,
                output_attentions,
            )
            attention_output = cross_attention_outputs[0]
            outputs  =  outputs  +  cross_attention_outputs[1:]    #  add
cross attentions if we output attention weights

        layer_output = apply_chunking_to_forward(
            self.feed_forward_chunk,      self.chunk_size_feed_forward,
self.seq_len_dim, attention_output
        )
        outputs = (layer_output,) + outputs
        return outputs

    def feed_forward_chunk(self, attention_output):
        intermediate_output = self.intermediate(attention_output)
        layer_output         =         self.output(intermediate_output,
attention_output)
        return layer_output


class BertEncoder(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.config = config
        self.layer   =   nn.ModuleList([BertLayer(config)   for   _   in
range(config.num_hidden_layers)])

    def forward(
        self,
```

```
        hidden_states,
        attention_mask=None,
        head_mask=None,
        encoder_hidden_states=None,
        encoder_attention_mask=None,
        output_attentions=False,
        output_hidden_states=False,
        return_dict=True,
    ):
        all_hidden_states = () if output_hidden_states else None
        all_self_attentions = () if output_attentions else None
        all_cross_attentions    =    ()    if    output_attentions    and
self.config.add_cross_attention else None
        for i, layer_module in enumerate(self.layer):
            if output_hidden_states:
                all_hidden_states      =      all_hidden_states      +
(hidden_states,)

            layer_head_mask  =  head_mask[i]  if  head_mask  is  not  None
else None

            if getattr(self.config, "gradient_checkpointing", False):

                def create_custom_forward(module):
                    def custom_forward(*inputs):
                        return module(*inputs, output_attentions)

                    return custom_forward

                layer_outputs = torch.utils.checkpoint.checkpoint(
                    create_custom_forward(layer_module),

                  hidden_states,
                    attention_mask,
                    layer_head_mask,
                    encoder_hidden_states,
                    encoder_attention_mask,
                )
            else:
                layer_outputs = layer_module(
                    hidden_states,
                    attention_mask,
                    layer_head_mask,
                    encoder_hidden_states,
                    encoder_attention_mask,
                    output_attentions,
                )
            hidden_states = layer_outputs[0]
            if output_attentions:
                all_self_attentions     =     all_self_attentions     +
(layer_outputs[1],)
                if self.config.add_cross_attention:
```

```
                    all_cross_attentions = all_cross_attentions +
(layer_outputs[2],)

        if output_hidden_states:
            all_hidden_states = all_hidden_states + (hidden_states,)

        if not return_dict:
            return tuple(
                v
                for v in [hidden_states, all_hidden_states,
all_self_attentions, all_cross_attentions]
                if v is not None
            )
        return BaseModelOutputWithCrossAttentions(
            last_hidden_state=hidden_states,
            hidden_states=all_hidden_states,
            attentions=all_self_attentions,
            cross_attentions=all_cross_attentions,)

class BertPooler(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        self.activation = nn.Tanh()

    def forward(self, hidden_states):
        # We "pool" the model by simply taking the hidden state
corresponding
        # to the first token.
        first_token_tensor = hidden_states[:, 0]
        pooled_output = self.dense(first_token_tensor)
        pooled_output = self.activation(pooled_output)
        return pooled_output


class BertPredictionHeadTransform(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.dense = nn.Linear(config.hidden_size, config.hidden_size)
        if isinstance(config.hidden_act, str):
            self.transform_act_fn = ACT2FN[config.hidden_act]
        else:
            self.transform_act_fn = config.hidden_act
        self.LayerNorm = nn.LayerNorm(config.hidden_size,
eps=config.layer_norm_eps)

    def forward(self, hidden_states):
        hidden_states = self.dense(hidden_states)
        hidden_states = self.transform_act_fn(hidden_states)
        hidden_states = self.LayerNorm(hidden_states)
        return hidden_states
```

```
class BertLMPredictionHead(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.transform = BertPredictionHeadTransform(config)


     # The output weights are the same as the input embeddings, but
there is
        # an output-only bias for each token.
        self.decoder = nn.Linear(config.hidden_size, config.vocab_size,
bias=False)

        self.bias = nn.Parameter(torch.zeros(config.vocab_size))

        # Need a link between the two variables so that the bias is
correctly resized with `resize_token_embeddings`
        self.decoder.bias = self.bias

    def forward(self, hidden_states):
        hidden_states = self.transform(hidden_states)
        hidden_states = self.decoder(hidden_states)
        return hidden_states


class BertOnlyMLMHead(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.predictions = BertLMPredictionHead(config)

    def forward(self, sequence_output):
        prediction_scores = self.predictions(sequence_output)
        return prediction_scores


class BertOnlyNSPHead(nn.Module):
    def __init__(self, config):
        super().__init__()
        self.seq_relationship = nn.Linear(config.hidden_size, 2)

    def forward(self, pooled_output):
        seq_relationship_score = self.seq_relationship(pooled_output)
        return seq_relationship_score


class BertPreTrainingHeads(nn.Module):

   def __init__(self, config):
        super().__init__()
        self.predictions = BertLMPredictionHead(config)
        self.seq_relationship = nn.Linear(config.hidden_size, 2)

    def forward(self, sequence_output, pooled_output):
        prediction_scores = self.predictions(sequence_output)
```

```
        seq_relationship_score = self.seq_relationship(pooled_output)
        return prediction_scores, seq_relationship_score


class BertPreTrainedModel(PreTrainedModel):
    """
    An abstract class to handle weights initialization and a simple
interface for downloading and loading pretrained
    models.
    """

    config_class = BertConfig
    load_tf_weights = load_tf_weights_in_bert
    base_model_prefix = "bert"
    _keys_to_ignore_on_load_missing = [r"position_ids"]

    def _init_weights(self, module):
        """ Initialize the weights """
        if isinstance(module, (nn.Linear, nn.Embedding)):
            #  Slightly different from the TF version which uses
truncated_normal for initialization
            # cf https://github.com/pytorch/pytorch/pull/5617
            module.weight.data.normal_(mean=0.0,
std=self.config.initializer_range)
        elif isinstance(module, nn.LayerNorm):
            module.bias.data.zero_()
            module.weight.data.fill_(1.0)
        if isinstance(module, nn.Linear) and module.bias is not None:
            module.bias.data.zero_()



@dataclass
class BertForPreTrainingOutput(ModelOutput):
    """
    Output type of :class:`~transformers.BertForPreTraining`.

    Args:
        loss  (`optional`,  returned  when  ``labels``  is  provided,
``torch.FloatTensor`` of shape :obj:`(1,)`):
            Total loss as the sum of the masked language modeling loss
and the next sequence prediction
            (classification) loss.
        prediction_logits          (:obj:`torch.FloatTensor`          of
shape :obj:`(batch_size, sequence_length, config.vocab_size)`):
            Prediction scores of the language modeling head (scores
for each vocabulary token before SoftMax).
        seq_relationship_logits      (:obj:`torch.FloatTensor`      of
shape :obj:`(batch_size, 2)`):
            Prediction  scores  of  the  next  sequence  prediction
(classification) head (scores of True/False continuation
            before SoftMax).
```

```
        hidden_states   (:obj:`tuple(torch.FloatTensor)`,   `optional`,
returned   when   ``output_hidden_states=True``   is   passed   or   when
``config.output_hidden_states=True``):
            Tuple of :obj:`torch.FloatTensor` (one for the output of
the embeddings + one for the output of each layer)
            of shape :obj:`(batch_size, sequence_length, hidden_size)`.

            Hidden-states of the model at the output of each layer
plus the initial embedding outputs.
        attentions    (:obj:`tuple(torch.FloatTensor)`,    `optional`,
returned   when   ``output_attentions=True``   is   passed   or   when
``config.output_attentions=True``):
            Tuple of :obj:`torch.FloatTensor` (one for each layer) of
shape :obj:`(batch_size, num_heads,
            sequence_length, sequence_length)`.

            Attentions weights after the attention softmax, used to
compute the weighted average in the self-attention

            heads.
    """

    loss: Optional[torch.FloatTensor] = None
    prediction_logits: torch.FloatTensor = None
    seq_relationship_logits: torch.FloatTensor = None
    hidden_states: Optional[Tuple[torch.FloatTensor]] = None
    attentions: Optional[Tuple[torch.FloatTensor]] = None


BERT_START_DOCSTRING = r"""

    This model inherits from :class:`~transformers.PreTrainedModel`.
Check the superclass documentation for the generic
    methods the library implements for all its model (such as
downloading or saving, resizing the input embeddings,
    pruning heads etc.)

    This    model    is    also    a    PyTorch    `torch.nn.Module
<https://pytorch.org/docs/stable/nn.html#torch.nn.Module>`__
    subclass. Use it as a regular PyTorch Module and refer to the
PyTorch documentation for all matter related to
    general usage and behavior.

    Parameters:
        config        (:class:`~transformers.BertConfig`):        Model
configuration class with all the parameters of the model.
            Initializing with a config file does not load the weights
associated with the model, only the
            configuration.                  Check                  out
the  :meth:`~transformers.PreTrainedModel.from_pretrained`  method  to
load the model
            weights.
"""
```

```
BERT_INPUTS_DOCSTRING = r"""
    Args:
        input_ids (:obj:`torch.LongTensor` of shape :obj:`({0})`):

            Indices of input sequence tokens in the vocabulary.

            Indices             can             be             obtained
using :class:`~transformers.BertTokenizer`. See
            :meth:`transformers.PreTrainedTokenizer.encode`
and :meth:`transformers.PreTrainedTokenizer.__call__` for
            details.

            `What are input IDs? <../glossary.html#input-ids>`__
        attention_mask (:obj:`torch.FloatTensor` of shape :obj:`({0})`,
`optional`):
            Mask  to  avoid  performing  attention  on  padding  token
indices. Mask values selected in ``[0, 1]``:

            - 1 for tokens that are **not masked**,
            - 0 for tokens that are **masked**.

            `What  are  attention  masks?  <../glossary.html#attention-
mask>`__
        token_type_ids (:obj:`torch.LongTensor` of shape :obj:`({0})`,
`optional`):
            Segment  token  indices  to  indicate  first  and  second
portions of the inputs. Indices are selected in ``[0,
            1]``:

            - 0 corresponds to a `sentence A` token,
            - 1 corresponds to a `sentence B` token.

            `What  are  token  type  IDs?  <../glossary.html#token-type-
ids>`_
        position_ids  (:obj:`torch.LongTensor`  of  shape  :obj:`({0})`,
`optional`):
            Indices of positions of each input sequence tokens in the
position embeddings. Selected in the range ``[0,
            config.max_position_embeddings - 1]``.

            `What are position IDs? <../glossary.html#position-ids>`_
        head_mask              (:obj:`torch.FloatTensor`              of
shape    :obj:`(num_heads,)`    or    :obj:`(num_layers,    num_heads)`,
`optional`):


            Mask  to  nullify  selected  heads  of  the  self-attention
modules. Mask values selected in ``[0, 1]``:

            - 1 indicates the head is **not masked**,
            - 0 indicates the head is **masked**.
```

```
        inputs_embeds (:obj:`torch.FloatTensor` of shape :obj:`({0},
hidden_size)`, `optional`):
            Optionally, instead of passing :obj:`input_ids` you can
choose to directly pass an embedded representation.
            This is useful if you want more control over how to
convert :obj:`input_ids` indices into associated
            vectors than the model's internal embedding lookup matrix.
        output_attentions (:obj:`bool`, `optional`):
            Whether or not to return the attentions tensors of all
attention layers. See ``attentions`` under returned
            tensors for more detail.
        output_hidden_states (:obj:`bool`, `optional`):
            Whether or not to return the hidden states of all layers.
See ``hidden_states`` under returned tensors for
            more detail.
        return_dict (:obj:`bool`, `optional`):
            Whether or not to return
a :class:`~transformers.file_utils.ModelOutput` instead of a plain
tuple.
"""


@add_start_docstrings(
    "The bare Bert Model transformer outputting raw hidden-states
without any specific head on top.",
    BERT_START_DOCSTRING,
)
class BertModel(BertPreTrainedModel):
    """

    The model can behave as an encoder (with only self-attention) as
well as a decoder, in which case a layer of


   cross-attention is added between the self-attention layers,
following the architecture described in `Attention is
    all you need <https://arxiv.org/abs/1706.03762>`__ by Ashish
Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit,
    Llion Jones, Aidan N. Gomez, Lukasz Kaiser and Illia Polosukhin.

    To behave as an decoder the model needs to be initialized with
the :obj:`is_decoder` argument of the configuration
    set to :obj:`True`. To be used in a Seq2Seq model, the model needs
to initialized with both :obj:`is_decoder`
    argument and :obj:`add_cross_attention` set to :obj:`True`;
an :obj:`encoder_hidden_states` is then expected as an
    input to the forward pass.
    """

    def __init__(self, config, add_pooling_layer=True):
        super().__init__(config)
        self.config = config
```

```
        self.embeddings = BertEmbeddings(config)
        self.encoder = BertEncoder(config)

        self.pooler = BertPooler(config) if add_pooling_layer else
None

        self.init_weights()

    def get_input_embeddings(self):
        return self.embeddings.word_embeddings

    def set_input_embeddings(self, value):
        self.embeddings.word_embeddings = value

    def _prune_heads(self, heads_to_prune):
        """
        Prunes heads of the model. heads_to_prune: dict of {layer_num:
list of heads to prune in this layer} See base
        class PreTrainedModel
        """

       for layer, heads in heads_to_prune.items():
            self.encoder.layer[layer].attention.prune_heads(heads)


@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, sequence_length"))
    @add_code_sample_docstrings(
        tokenizer_class=_TOKENIZER_FOR_DOC,
        checkpoint="bert-base-uncased",
        output_type=BaseModelOutputWithPoolingAndCrossAttentions,
        config_class=_CONFIG_FOR_DOC,
    )
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        encoder_hidden_states=None,
        encoder_attention_mask=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):
        r"""
        encoder_hidden_states          (:obj:`torch.FloatTensor`     of
shape :obj:`(batch_size, sequence_length, hidden_size)`, `optional`):
            Sequence of hidden-states at the output of the last layer
of the encoder. Used in the cross-attention if
            the model is configured as a decoder.
```

```
        encoder_attention_mask       (:obj:`torch.FloatTensor`       of
shape :obj:`(batch_size, sequence_length)`, `optional`):
            Mask to avoid performing attention on the padding token
indices of the encoder input. This mask is used in
            the cross-attention if the model is configured as a
decoder. Mask values selected in ``[0, 1]``:


            - 1 for tokens that are **not masked**,
            - 0 for tokens that are **masked**.
        """
        output_attentions = output_attentions if output_attentions is
not None else self.config.output_attentions
        output_hidden_states = (
            output_hidden_states if output_hidden_states is not None
else self.config.output_hidden_states
        )
        return_dict = return_dict if return_dict is not None else
self.config.use_return_dict

        if input_ids is not None and inputs_embeds is not None:
            raise ValueError("You cannot specify both input_ids and
inputs_embeds at the same time")
        elif input_ids is not None:
            input_shape = input_ids.size()
        elif inputs_embeds is not None:
            input_shape = inputs_embeds.size()[:-1]
        else:
            raise ValueError("You have to specify either input_ids or
inputs_embeds")

        device = input_ids.device if input_ids is not None else
inputs_embeds.device

        if attention_mask is None:
            attention_mask = torch.ones(input_shape, device=device)
        if token_type_ids is None:
            token_type_ids = torch.zeros(input_shape, dtype=torch.long,
device=device)

        # We can provide a self-attention mask of dimensions
[batch_size, from_seq_length, to_seq_length]
        # ourselves in which case we just need to make it
broadcastable to all heads.


       extended_attention_mask:             torch.Tensor             =
self.get_extended_attention_mask(attention_mask, input_shape, device)

        # If a 2D or 3D attention mask is provided for the cross-
attention
        # we need to make broadcastable to [batch_size, num_heads,
seq_length, seq_length]
```

```
        if self.config.is_decoder and encoder_hidden_states is not
None:
            encoder_batch_size,    encoder_sequence_length,    _    =
encoder_hidden_states.size()
            encoder_hidden_shape        =        (encoder_batch_size,
encoder_sequence_length)
            if encoder_attention_mask is None:
                encoder_attention_mask                                =
torch.ones(encoder_hidden_shape, device=device)
            encoder_extended_attention_mask                           =
self.invert_attention_mask(encoder_attention_mask)
        else:
            encoder_extended_attention_mask = None

        # Prepare head mask if needed
        # 1.0 in head_mask indicate we keep the head
        # attention_probs has shape bsz x n_heads x N x N
        # input head_mask has shape [num_heads] or [num_hidden_layers
x num_heads]
        # and head_mask is converted to shape [num_hidden_layers x
batch x num_heads x seq_length x seq_length]
        head_mask           =           self.get_head_mask(head_mask,
self.config.num_hidden_layers)

        embedding_output = self.embeddings(
            input_ids=input_ids,            position_ids=position_ids,
token_type_ids=token_type_ids, inputs_embeds=inputs_embeds
        )
        encoder_outputs = self.encoder(
            embedding_output,
            attention_mask=extended_attention_mask,
            head_mask=head_mask,
            encoder_hidden_states=encoder_hidden_states,

            encoder_attention_mask=encoder_extended_attention_mask,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )
        sequence_output = encoder_outputs[0]
        pooled_output = self.pooler(sequence_output) if self.pooler is
not None else None

        if not return_dict:
            return       (sequence_output,       pooled_output)       +
encoder_outputs[1:]

        return BaseModelOutputWithPoolingAndCrossAttentions(
            last_hidden_state=sequence_output,
            pooler_output=pooled_output,
            hidden_states=encoder_outputs.hidden_states,
            attentions=encoder_outputs.attentions,
            cross_attentions=encoder_outputs.cross_attentions,
```

```
        )


@add_start_docstrings(
    """
    Bert Model with two heads on top as done during the pretraining: a
`masked language modeling` head and a `next
    sentence prediction (classification)` head.
    """,
    BERT_START_DOCSTRING,
)
class BertForPreTraining(BertPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)

        self.bert = BertModel(config)
        self.cls = BertPreTrainingHeads(config)

        self.init_weights()


    def get_output_embeddings(self):
        return self.cls.predictions.decoder

    def set_output_embeddings(self, new_embeddings):
        self.cls.predictions.decoder = new_embeddings


@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, sequence_length"))
    @replace_return_docstrings(output_type=BertForPreTrainingOutput,
config_class=_CONFIG_FOR_DOC)
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        labels=None,
        next_sentence_label=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):
        r"""
        labels   (:obj:`torch.LongTensor`   of   shape   ``(batch_size,
sequence_length)``, `optional`):
            Labels for computing the masked language modeling loss.
Indices should be in ``[-100, 0, ...,
            config.vocab_size]`` (see ``input_ids`` docstring) Tokens
with indices set to ``-100`` are ignored
```

```
            (masked), the loss is only computed for the tokens with
labels in ``[0, ..., config.vocab_size]``
        next_sentence_label     (``torch.LongTensor``     of     shape
``(batch_size,)``, `optional`):
            Labels for computing the next sequence prediction
(classification) loss. Input should be a sequence pair

           (see :obj:`input_ids` docstring) Indices should be in ``[0,
1]``:

            - 0 indicates sequence B is a continuation of sequence A,
            - 1 indicates sequence B is a random sequence.
        kwargs (:obj:`Dict[str, any]`, optional, defaults to `{}`):
            Used to hide legacy arguments that have been deprecated.

        Returns:

        Example::

            >>>     from     transformers     import     BertTokenizer,
BertForPreTraining
            >>> import torch

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

            >>>   inputs   =   tokenizer("Hello,   my   dog   is   cute",
return_tensors="pt")
            >>> outputs = model(**inputs)

            >>> prediction_logits = outputs.prediction_logits
            >>>                seq_relationship_logits                =
outputs.seq_relationship_logits
        """
        return_dict = return_dict if return_dict is not None else
self.config.use_return_dict

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,

        )

        sequence_output, pooled_output = outputs[:2]
```

```
        prediction_scores,           seq_relationship_score           =
self.cls(sequence_output, pooled_output)

        total_loss = None
        if labels is not None and next_sentence_label is not None:
            loss_fct = CrossEntropyLoss()
            masked_lm_loss    =    loss_fct(prediction_scores.view(-1,
self.config.vocab_size), labels.view(-1))
            next_sentence_loss                                          =
loss_fct(seq_relationship_score.view(-1,                                2),
next_sentence_label.view(-1))
            total_loss = masked_lm_loss + next_sentence_loss

        if not return_dict:
            output  =  (prediction_scores,  seq_relationship_score)  +
outputs[2:]
            return ((total_loss,) + output) if total_loss is not None
else output

        return BertForPreTrainingOutput(
            loss=total_loss,
            prediction_logits=prediction_scores,
            seq_relationship_logits=seq_relationship_score,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


@add_start_docstrings(
    """Bert Model with a `language modeling` head on top for CLM fine-
tuning. """, BERT_START_DOCSTRING
)
class BertLMHeadModel(BertPreTrainedModel):

    _keys_to_ignore_on_load_unexpected = [r"pooler"]
    _keys_to_ignore_on_load_missing         =         [r"position_ids",
r"predictions.decoder.bias"]

    def __init__(self, config):
        super().__init__(config)

        if not config.is_decoder:
            logger.warning("If you want to use `BertLMHeadModel` as a
standalone, add `is_decoder=True.`")

        self.bert = BertModel(config, add_pooling_layer=False)
        self.cls = BertOnlyMLMHead(config)

        self.init_weights()

    def get_output_embeddings(self):
        return self.cls.predictions.decoder
```

```
    def set_output_embeddings(self, new_embeddings):
        self.cls.predictions.decoder = new_embeddings


@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, sequence_length"))

@replace_return_docstrings(output_type=CausalLMOutputWithCrossAttentio
ns, config_class=_CONFIG_FOR_DOC)
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        encoder_hidden_states=None,
        encoder_attention_mask=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):

       r"""
        encoder_hidden_states          (:obj:`torch.FloatTensor`     of
shape :obj:`(batch_size, sequence_length, hidden_size)`, `optional`):
            Sequence of hidden-states at the output of the last layer
of the encoder. Used in the cross-attention if
            the model is configured as a decoder.
        encoder_attention_mask       (:obj:`torch.FloatTensor`       of
shape :obj:`(batch_size, sequence_length)`, `optional`):
            Mask to avoid performing attention on the padding token
indices of the encoder input. This mask is used in
            the cross-attention if the model is configured as a
decoder. Mask values selected in ``[0, 1]``:

            - 1 for tokens that are **not masked**,
            - 0 for tokens that are **masked**.
        labels  (:obj:`torch.LongTensor` of shape :obj:`(batch_size,
sequence_length)`, `optional`):
            Labels for computing the left-to-right language modeling
loss (next word prediction). Indices should be in
            ``[-100, 0, ..., config.vocab_size]`` (see ``input_ids``
docstring) Tokens with indices set to ``-100`` are
            ignored (masked), the loss is only computed for the tokens
with labels n ``[0, ..., config.vocab_size]``

        Returns:

        Example::
```

```
            >>>     from     transformers     import     BertTokenizer,
BertLMHeadModel, BertConfig
            >>> import torch

            >>> tokenizer = BertTokenizer.from_pretrained('bert-base-
cased')
            >>> config = BertConfig.from_pretrained("bert-base-cased")
            >>> config.is_decoder = True
            >>>  model  =  BertLMHeadModel.from_pretrained('bert-base-
cased', config=config)


          >>>  inputs  =  tokenizer("Hello,  my  dog  is  cute",
return_tensors="pt")
            >>> outputs = model(**inputs)

            >>> prediction_logits = outputs.logits
        """
        return_dict = return_dict if return_dict is not None else
self.config.use_return_dict

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            encoder_hidden_states=encoder_hidden_states,
            encoder_attention_mask=encoder_attention_mask,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )

        sequence_output = outputs[0]
        prediction_scores = self.cls(sequence_output)

        lm_loss = None
        if labels is not None:
            # we are doing next-token prediction; shift prediction
scores and input ids by one
            shifted_prediction_scores   =   prediction_scores[:,   :-
1, :].contiguous()
            labels = labels[:, 1:].contiguous()
            loss_fct = CrossEntropyLoss()
            lm_loss    =    loss_fct(shifted_prediction_scores.view(-1,
self.config.vocab_size), labels.view(-1))

        if not return_dict:

           output = (prediction_scores,) + outputs[2:]
```

```
            return ((lm_loss,) + output) if lm_loss is not None else
output

        return CausalLMOutputWithCrossAttentions(
            loss=lm_loss,
            logits=prediction_scores,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
            cross_attentions=outputs.cross_attentions,
        )

    def          prepare_inputs_for_generation(self,          input_ids,
attention_mask=None, **model_kwargs):
        input_shape = input_ids.shape

        # if model is used as a decoder in encoder-decoder model, the
decoder attention mask is created on the fly
        if attention_mask is None:
            attention_mask = input_ids.new_ones(input_shape)

        return      {"input_ids":      input_ids,      "attention_mask":
attention_mask}


@add_start_docstrings("""Bert Model with a `language modeling` head on
top. """, BERT_START_DOCSTRING)
class BertForMaskedLM(BertPreTrainedModel):

    _keys_to_ignore_on_load_unexpected = [r"pooler"]
    _keys_to_ignore_on_load_missing         =         [r"position_ids",
r"predictions.decoder.bias"]

    def __init__(self, config):
        super().__init__(config)

        if config.is_decoder:
            logger.warning(
                "If  you  want  to  use  `BertForMaskedLM`  make  sure
`config.is_decoder=False` for "

                "bi-directional self-attention."
            )

        self.bert = BertModel(config, add_pooling_layer=False)
        self.cls = BertOnlyMLMHead(config)

        self.init_weights()

    def get_output_embeddings(self):
        return self.cls.predictions.decoder

    def set_output_embeddings(self, new_embeddings):
        self.cls.predictions.decoder = new_embeddings
```

```
@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, sequence_length"))
    @add_code_sample_docstrings(
        tokenizer_class=_TOKENIZER_FOR_DOC,
        checkpoint="bert-base-uncased",
        output_type=MaskedLMOutput,
        config_class=_CONFIG_FOR_DOC,
    )
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        encoder_hidden_states=None,
        encoder_attention_mask=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):

        r"""
        labels (:obj:`torch.LongTensor` of shape :obj:`(batch_size,
sequence_length)`, `optional`):
            Labels for computing the masked language modeling loss.
Indices should be in ``[-100, 0, ...,
            config.vocab_size]`` (see ``input_ids`` docstring) Tokens
with indices set to ``-100`` are ignored
            (masked), the loss is only computed for the tokens with
labels in ``[0, ..., config.vocab_size]``
        """

        return_dict = return_dict if return_dict is not None else
self.config.use_return_dict

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            encoder_hidden_states=encoder_hidden_states,
            encoder_attention_mask=encoder_attention_mask,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )
```

```
        sequence_output = outputs[0]
        prediction_scores = self.cls(sequence_output)

        masked_lm_loss = None
        if labels is not None:
            loss_fct = CrossEntropyLoss()  # -100 index = padding
token
            masked_lm_loss    =    loss_fct(prediction_scores.view(-1,
self.config.vocab_size), labels.view(-1))

        if not return_dict:

            output = (prediction_scores,) + outputs[2:]
            return ((masked_lm_loss,) + output) if masked_lm_loss is
not None else output

        return MaskedLMOutput(
            loss=masked_lm_loss,
            logits=prediction_scores,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )

    def         prepare_inputs_for_generation(self,         input_ids,
attention_mask=None, **model_kwargs):
        input_shape = input_ids.shape
        effective_batch_size = input_shape[0]

        #  add a dummy token
        assert self.config.pad_token_id is not None, "The PAD token
should be defined for generation"
        attention_mask          =          torch.cat([attention_mask,
attention_mask.new_zeros((attention_mask.shape[0], 1))], dim=-1)
        dummy_token = torch.full(
            (effective_batch_size,    1),    self.config.pad_token_id,
dtype=torch.long, device=input_ids.device
        )
        input_ids = torch.cat([input_ids, dummy_token], dim=1)

        return     {"input_ids":     input_ids,     "attention_mask":
attention_mask}


@add_start_docstrings(
    """Bert Model with a `next sentence prediction (classification)`
head on top. """,
    BERT_START_DOCSTRING,
)
class BertForNextSentencePrediction(BertPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
```

```
        self.bert = BertModel(config)
        self.cls = BertOnlyNSPHead(config)

        self.init_weights()


@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, sequence_length"))
    @replace_return_docstrings(output_type=NextSentencePredictorOutput,
config_class=_CONFIG_FOR_DOC)
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
        **kwargs
    ):
        r"""
        labels (:obj:`torch.LongTensor` of shape :obj:`(batch_size,)`,
`optional`):
            Labels for computing the next sequence prediction
(classification) loss. Input should be a sequence pair
            (see ``input_ids`` docstring). Indices should be in ``[0,
1]``:

            - 0 indicates sequence B is a continuation of sequence A,
            - 1 indicates sequence B is a random sequence.

        Returns:

        Example::


           >>>     from     transformers     import     BertTokenizer,
BertForNextSentencePrediction
            >>> import torch

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

            >>> prompt = "In Italy, pizza served in formal settings,
such as at a restaurant, is presented unsliced."
            >>> next_sentence = "The sky is blue due to the shorter
wavelength of blue light."
```

```
            >>>    encoding    =    tokenizer(prompt,    next_sentence,
return_tensors='pt')

            >>>          outputs          =          model(**encoding,
labels=torch.LongTensor([1]))
            >>> logits = outputs.logits
            >>> assert logits[0, 0] < logits[0, 1] # next sentence was
random
        """

        if "next_sentence_label" in kwargs:
            warnings.warn(
                "The `next_sentence_label` argument is deprecated and
will be removed in a future version, use `labels` instead.",
                FutureWarning,
            )
            labels = kwargs.pop("next_sentence_label")

        return_dict  =  return_dict  if  return_dict  is  not  None  else
self.config.use_return_dict

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,

            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )

        pooled_output = outputs[1]

        seq_relationship_scores = self.cls(pooled_output)

        next_sentence_loss = None
        if labels is not None:
            loss_fct = CrossEntropyLoss()
            next_sentence_loss                                          =
loss_fct(seq_relationship_scores.view(-1, 2), labels.view(-1))

        if not return_dict:
            output = (seq_relationship_scores,) + outputs[2:]
            return     ((next_sentence_loss,)     +     output)     if
next_sentence_loss is not None else output

        return NextSentencePredictorOutput(
            loss=next_sentence_loss,
            logits=seq_relationship_scores,
            hidden_states=outputs.hidden_states,
```

```
            attentions=outputs.attentions,
        )


@add_start_docstrings(
    """
    Bert Model transformer with a sequence classification/regression
head on top (a linear layer on top of the pooled
    output) e.g. for GLUE tasks.
    """,
    BERT_START_DOCSTRING,
)
class BertForSequenceClassification(BertPreTrainedModel):
    def __init__(self, config):

        super().__init__(config)
        self.num_labels = config.num_labels

        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier         =         nn.Linear(config.hidden_size,
config.num_labels)

        self.init_weights()


@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, sequence_length"))
    @add_code_sample_docstrings(
        tokenizer_class=_TOKENIZER_FOR_DOC,
        checkpoint="bert-base-uncased",
        output_type=SequenceClassifierOutput,
        config_class=_CONFIG_FOR_DOC,
    )
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):
        r"""
        labels (:obj:`torch.LongTensor` of shape :obj:`(batch_size,)`,
`optional`):
            Labels       for       computing       the       sequence
classification/regression loss. Indices should be in :obj:`[0, ...,
```

```
            config.num_labels - 1]`. If :obj:`config.num_labels == 1`
a regression loss is computed (Mean-Square loss),

           If :obj:`config.num_labels > 1` a classification loss is
computed (Cross-Entropy).
        """
        return_dict = return_dict if return_dict is not None else
self.config.use_return_dict

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )

        pooled_output = outputs[1]

        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)

        loss = None
        if labels is not None:
            if self.num_labels == 1:
                #  We are doing regression
                loss_fct = MSELoss()
                loss = loss_fct(logits.view(-1), labels.view(-1))
            else:
                loss_fct = CrossEntropyLoss()
                loss   =   loss_fct(logits.view(-1,   self.num_labels),
labels.view(-1))

        if not return_dict:
            output = (logits,) + outputs[2:]
            return ((loss,) + output) if loss is not None else output


        return SequenceClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


@add_start_docstrings(
    """
```

```
    Bert Model with a multiple choice classification head on top (a
linear layer on top of the pooled output and a
    softmax) e.g. for RocStories/SWAG tasks.
    """,
    BERT_START_DOCSTRING,
)
class BertForMultipleChoice(BertPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)

        self.bert = BertModel(config)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier = nn.Linear(config.hidden_size, 1)

        self.init_weights()


@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, num_choices, sequence_length"))
    @add_code_sample_docstrings(
        tokenizer_class=_TOKENIZER_FOR_DOC,
        checkpoint="bert-base-uncased",
        output_type=MultipleChoiceModelOutput,
        config_class=_CONFIG_FOR_DOC,
    )
    def forward(
        self,
        input_ids=None,

        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        labels=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):
        r"""
        labels (:obj:`torch.LongTensor` of shape :obj:`(batch_size,)`,
`optional`):
            Labels for computing the multiple choice classification
loss. Indices should be in ``[0, ...,
            num_choices-1]`` where :obj:`num_choices` is the size of
the second dimension of the input tensors. (See
            :obj:`input_ids` above)
        """
        return_dict = return_dict if return_dict is not None else
self.config.use_return_dict
        num_choices = input_ids.shape[1] if input_ids is not None else
inputs_embeds.shape[1]
```

```
        input_ids    =    input_ids.view(-1,    input_ids.size(-1))    if
input_ids is not None else None
        attention_mask = attention_mask.view(-1, attention_mask.size(-
1)) if attention_mask is not None else None
        token_type_ids = token_type_ids.view(-1, token_type_ids.size(-
1)) if token_type_ids is not None else None
        position_ids = position_ids.view(-1, position_ids.size(-1)) if
position_ids is not None else None
        inputs_embeds = (
            inputs_embeds.view(-1,              inputs_embeds.size(-2),
inputs_embeds.size(-1))
            if inputs_embeds is not None
            else None
        )

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )

        pooled_output = outputs[1]

        pooled_output = self.dropout(pooled_output)
        logits = self.classifier(pooled_output)
        reshaped_logits = logits.view(-1, num_choices)

        loss = None
        if labels is not None:
            loss_fct = CrossEntropyLoss()
            loss = loss_fct(reshaped_logits, labels)

        if not return_dict:
            output = (reshaped_logits,) + outputs[2:]
            return ((loss,) + output) if loss is not None else output

        return MultipleChoiceModelOutput(
            loss=loss,
            logits=reshaped_logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


@add_start_docstrings(
    """
```

```
    Bert Model with a token classification head on top (a linear layer
on top of the hidden-states output) e.g. for
    Named-Entity-Recognition (NER) tasks.
    """,
    BERT_START_DOCSTRING,
)
class BertForTokenClassification(BertPreTrainedModel):

    _keys_to_ignore_on_load_unexpected = [r"pooler"]

    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels

        self.bert = BertModel(config, add_pooling_layer=False)
        self.dropout = nn.Dropout(config.hidden_dropout_prob)
        self.classifier         =         nn.Linear(config.hidden_size,
config.num_labels)

        self.init_weights()


@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, sequence_length"))
    @add_code_sample_docstrings(
        tokenizer_class=_TOKENIZER_FOR_DOC,
        checkpoint="bert-base-uncased",
        output_type=TokenClassifierOutput,
        config_class=_CONFIG_FOR_DOC,
    )
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        labels=None,

        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):
        r"""
        labels  (:obj:`torch.LongTensor`  of  shape  :obj:`(batch_size,
sequence_length)`, `optional`):
            Labels  for  computing  the  token  classification  loss.
Indices should be in ``[0, ..., config.num_labels -
            1]``.
        """
```

```
        return_dict = return_dict if return_dict is not None else
self.config.use_return_dict

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )

        sequence_output = outputs[0]

        sequence_output = self.dropout(sequence_output)
        logits = self.classifier(sequence_output)

        loss = None
        if labels is not None:
            loss_fct = CrossEntropyLoss()
            # Only keep active parts of the loss
            if attention_mask is not None:
                active_loss = attention_mask.view(-1) == 1
                active_logits = logits.view(-1, self.num_labels)

                active_labels = torch.where(
                    active_loss,                          labels.view(-1),
torch.tensor(loss_fct.ignore_index).type_as(labels)
                )
                loss = loss_fct(active_logits, active_labels)
            else:
                loss   =   loss_fct(logits.view(-1,   self.num_labels),
labels.view(-1))

        if not return_dict:
            output = (logits,) + outputs[2:]
            return ((loss,) + output) if loss is not None else output

        return TokenClassifierOutput(
            loss=loss,
            logits=logits,
            hidden_states=outputs.hidden_states,
            attentions=outputs.attentions,
        )


@add_start_docstrings(
    """
    Bert Model with a span classification head on top for extractive
question-answering tasks like SQuAD (a linear
```

```
    layers on top of the hidden-states output to compute `span start
logits` and `span end logits`).
    """,
    BERT_START_DOCSTRING,
)
class BertForQuestionAnswering(BertPreTrainedModel):

    _keys_to_ignore_on_load_unexpected = [r"pooler"]

    def __init__(self, config):
        super().__init__(config)
        self.num_labels = config.num_labels


        self.bert = BertModel(config, add_pooling_layer=False)
        self.qa_outputs         =         nn.Linear(config.hidden_size,
config.num_labels)

        self.init_weights()


@add_start_docstrings_to_model_forward(BERT_INPUTS_DOCSTRING.format("b
atch_size, sequence_length"))
    @add_code_sample_docstrings(
        tokenizer_class=_TOKENIZER_FOR_DOC,
        checkpoint="bert-base-uncased",
        output_type=QuestionAnsweringModelOutput,
        config_class=_CONFIG_FOR_DOC,
    )
    def forward(
        self,
        input_ids=None,
        attention_mask=None,
        token_type_ids=None,
        position_ids=None,
        head_mask=None,
        inputs_embeds=None,
        start_positions=None,
        end_positions=None,
        output_attentions=None,
        output_hidden_states=None,
        return_dict=None,
    ):
        r"""
        start_positions           (:obj:`torch.LongTensor`           of
shape :obj:`(batch_size,)`, `optional`):
            Labels for position (index) of the start of the labelled
span for computing the token classification loss.
            Positions are clamped to the length of the sequence
(:obj:`sequence_length`). Position outside of the
            sequence are not taken into account for computing the loss.
        end_positions             (:obj:`torch.LongTensor`           of
shape :obj:`(batch_size,)`, `optional`):
```

```
            Labels for position (index) of the end of the labelled
span for computing the token classification loss.
            Positions are clamped to the length of the sequence
(:obj:`sequence_length`). Position outside of the
            sequence are not taken into account for computing the loss.
        """
        return_dict = return_dict if return_dict is not None else
self.config.use_return_dict

        outputs = self.bert(
            input_ids,
            attention_mask=attention_mask,
            token_type_ids=token_type_ids,
            position_ids=position_ids,
            head_mask=head_mask,
            inputs_embeds=inputs_embeds,
            output_attentions=output_attentions,
            output_hidden_states=output_hidden_states,
            return_dict=return_dict,
        )

        sequence_output = outputs[0]

        logits = self.qa_outputs(sequence_output)
        start_logits, end_logits = logits.split(1, dim=-1)
        start_logits = start_logits.squeeze(-1)
        end_logits = end_logits.squeeze(-1)

        total_loss = None
        if start_positions is not None and end_positions is not None:
            # If we are on multi-GPU, split add a dimension
            if len(start_positions.size()) > 1:
                start_positions = start_positions.squeeze(-1)
            if len(end_positions.size()) > 1:
                end_positions = end_positions.squeeze(-1)
            # sometimes the start/end positions are outside our model
inputs, we ignore these terms
            ignored_index = start_logits.size(1)

            start_positions.clamp_(0, ignored_index)
            end_positions.clamp_(0, ignored_index)

            loss_fct = CrossEntropyLoss(ignore_index=ignored_index)
            start_loss = loss_fct(start_logits, start_positions)
            end_loss = loss_fct(end_logits, end_positions)
            total_loss = (start_loss + end_loss) / 2

        if not return_dict:
            output = (start_logits, end_logits) + outputs[2:]
            return ((total_loss,) + output) if total_loss is not None
else output
```

```
return QuestionAnsweringModelOutput(
    loss=total_loss,
    start_logits=start_logits,
    end_logits=end_logits,
    hidden_states=outputs.hidden_states,
    attentions=outputs.attentions,
)
```