

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
Факультет кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютерних інформаційних технологій

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____ А.С. Савченко
« ____ » _____ 2021 р

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СПУПЕНЯ «БАКАЛАВР»

Тема: «Розумний чат-бот з технологією NLP»

Виконавець: студент УС-411 Марчук Владислав Сергійович
(студент, група, прізвище, ім'я, по батькові)

Керівник: д. т. н., професор Зіатдінов Юрій Кашафович
(науковий ступень, вчене звання, прізвище, ім'я, по батькові)

Нормоконтролер: ст. викл. Шевченко О.П.
(П.І.Б.) (підпис)

КИЇВ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет кібербезпеки, комп'ютерної та програмної інженерії

Кафедра комп'ютерних інформаційних технологій

Освітній ступінь: Бакалавр

Спеціальність, спеціалізація: 122 “Комп'ютерні науки”, “Інформаційні
управляючі системи та технології”

ЗАТВЕРДЖУЮ

Завідувач кафедри

А.С. Савченко

“ _____ ” _____ 2021 р.

ЗАВДАННЯ

на виконання дипломного проекту студента

Марчук Владислав Сергійович

(прізвище, ім'я, по батькові)

1. Тема проекту: «Розумний чат-бот з технологією NLP» затверджена наказом ректора № 636/ст. від 22.04.2021р.
2. Термін виконання роботи: з 10.05.2021 по 11.06.2021р.
3. Вихідні дані до роботи: інструкції для користувачів, літературні джерела з досліджуваної проблеми.
4. Зміст пояснювальної записки: вступ, аналітичний огляд і постановка завдання, розгляд завдання обробки природної мови, дослідження технологій та засобів, розробка програмного продукту, оцінка якості технології, висновки.
5. Перелік обов'язкового графічного матеріалу: загальний перелік існуючих мовних моделей та обробка природної мови створеним програмним продуктом. Використання структури проблематики питань обробки природної мови.

КАЛЕНДАРНИЙ ПЛАН

№	Етапи виконання дипломного проекту	Термін виконання етапів	Примітка
1	Аналіз літератури та джерел за темою дипломного проекту.	10.05.2021р. – 12.05.2021р.	
2	Розробка та затвердження плану дипломного проекту.	13.05.2021р.	
3	Проведення консультації з науковим керівником щодо створення першого розділу.	14.05.2021р.	
4	Аналітичний огляд і постановка задачі.	15.05.2021р. – 16.05.2021р.	
5	Порівняльний аналіз існуючих методів для реалізації обробки природної мови.	17.05.2021р. – 18.05.2021р.	
6	Тестування методів реалізації обробки природної мови.	19.06.2021р. – 23.05.2021р.	
7	Створення розумного чат-боту з технологією NLP.	24.05.2021р. – 29.06.2021р.	
8	Висновки та оформлення пояснювальної записки дипломного проекту.	30.06.2021р. – 31.06.2021р.	
9	Підписання необхідних документів у встановленому порядку.	01.06.2021р. – 04.06.2021р.	
10	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту.	05.06.2021р. – 11.06.2021р.	

Студент

(*Марчук В.С.*)

Керівник дипломного проекту

(*Зіатдінов Ю.К.*)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Розумний чат-бот з технологією NLP» містить: 96 сторінок, 21 рисунок, 35 літературних джерел.

Об'єкт дослідження: обробка природної мови за допомогою мовної моделі NLP.

Предмет дослідження: чат-бот з використанням обробки природної мови.

Мета проекту: створення програми для обробки тексту з використанням мовної моделі NLP.

Методи дослідження, технічні та програмні засоби: розробка програмних бібліотек, порівняльний аналіз, обробка літературних джерел.

Отримані результати та їх новизна: створено чат-бота для обробки повідомлень користувача та генерування тексту, подібного до людського мовлення. Продукт цілком адаптовано під мову користувача та зроблено запобігання раптовим програмним помилкам. У порівнянні отримано дані про поліпшення отриманих результатів. Програма може слугувати ядром у дослідженнях мовних моделей обробки природної мови.

ОБРОБКА ПРИРОДНОЇ МОВИ, ЧАТ-БОТ, ТРАНСФОРМЕР, МЕСЕНДЖЕР, МОВНА МОДЕЛЬ.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ.....	7
ВСТУП.....	8
РОЗДІЛ 1. ЗАВДАННЯ ОБРОБКИ ПРИРОДНОЇ МОВИ.....	10
1.1. Загальний опис обробки природної мови.....	10
1.2. Історія обробки природної мови.....	10
1.3. Методи обробки природної мови.....	13
1.4. Поширені завдання обробки природної мови.....	16
1.5. Трансформери в обробці природної мови.....	21
1.6. Чат-бот як реалізація моделі обробки природної мови.....	24
Висновки до розділу 1.....	26
РОЗДІЛ 2. ТЕХНОЛОГІЇ ТА ЗАСОБИ ОБРОБКИ ПРИРОДНОЇ МОВИ.....	27
2.1. Огляд мовних моделей для обробки природної мови.....	27
2.1.1. BERT: тренування глибоких двонаправлених трансформерів.....	27
2.1.2. GPT-2: багатозадачна неконтрольована модель.....	29
2.1.3. XLNet: узагальнена авторегресивна попередня підготовка.....	31
2.1.4. RoBERTa: надійно оптимізований підхід до підготовки BERT.....	33
2.1.5. ALBERT: легка BERT для самостійного вивчення мовних уявлень.....	34
2.1.6. StructBERT: включення мовних структур у попередню підготовку.....	35
2.1.7. T5: вивчення меж передавального навчання.....	36
2.1.8. GPT-3: модель що швидко навчається.....	38
2.1.9. ELECTRA: попередня підготовка кодерів тексту.....	40
2.1.10. DeBERTa: розширене декодування BERT.....	42
2.2. Огляд програм обміну повідомленнями для реалізації чат-боту.....	43
2.2.1. Facebook Messenger чат-бот.....	44
2.2.2. WhatsApp чат-бот.....	45
2.2.3. Slack чат-бот.....	46
2.2.4. WeChat чат-бот.....	47

2.2.5. Viber чат-бот.....	48
2.2.6. LINE чат-бот.....	50
2.2.7. Telegram чат-бот.....	51
Висновки до розділу 2.....	52
РОЗДІЛ 3. ТЕХНОЛОГІЯ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ.....	53
3.1. Опис роботи розумного чат-боту з технологією NLP.....	53
3.2. Демонстрація роботи розумного чат-боту з технологією NLP.....	57
Висновки до розділу 3.....	63
ВИСНОВКИ.....	64
СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ.....	65
ДОДАТКИ.....	69

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ ТА СКОРОЧЕНЬ

NLP	—	Natural Language Processing (обробка природної мови)
Токен	—	об'єкт, що створюється з лексеми в процесі лексичного аналізу
Трансформер	—	архітектура глибоких нейронних мереж
Чат-бот	—	система для обміну повідомленнями між людиною та комп'ютером
Мовна модель	—	інструмент, що аналізує людське мовлення для передбачення слів
BERT	—	мовна модель з двонаправленими представленнями кодерів від трансформерів
GPT-2	—	друге покоління генеративного попередньо навченого трансформеру
XLNet	—	авторегресивна мовна модель з взаємозалежністю між токенами
RoBERTa	—	сильно модифікована мовна модель BERT
ALBERT	—	полегшена мовна модель BERT
StructBERT	—	мовна модель BERT із включенням мовних структур у попередню підготовку
T5	—	трансформер з передавальним навчанням у форматі text-to-text
GPT-3	—	третє покоління генеративного попередньо навченого трансформеру
ELECTRA	—	модель з енкодером, що саморозвивається
DeBERTa	—	мовна модель з розширеним декодуванням BERT

ВСТУП

В наш час технології постійно розвиваються: розумний дім, електрокари, хмарні технології, 3D друк, постійно з'являються нові технології та тренди, деякі з яких здатні серйозно вплинути на всю сферу ІТ. Відносно недавно всі говорили лише про додатки, але останнім часом набирає обертів чат-ботів, яким пророкують велике майбутнє в комунікаціях та маркетингу.

Чому ж боти є актуальними, і навіть потрібними в наш час? Перш за все це дуже вигідно для бізнесу. Саме завдяки чат-ботам можна оптимізувати витрати та зменшити навантаження для компанії, їхня здатність виконувати понад 85% відсотків задач, які існують у взаємодії між організацією та клієнтом, чи всередині компанії для підвищення якості послуг.

Розумний чат-бот призначений для виконання одноманітної і рекурсивної роботи. У порівнянні з людьми, він має прискорену реакцію на процеси і не втомлюється від монотонних дій. Основні причини, для чого потрібні боти: для виконання розважальних, пізнавальних, комерційних дій. Великого поширення сьогодні отримали чат-боти, які розмовляють з людиною зрозумілою їй мовою. На питання клієнтів чат-бот дає заздалегідь запрограмовані відповіді. Такі утиліти активно використовуються в бізнесі: страхових компаніях, на інтернет-сторінках банків, інтернет-магазинах, лізингових компаніях та ін.

Об'єктом дослідження є розробка розумного чат-боту з використанням технології NLP – обробки природної мови.

Метою даної роботи є дослідження роботи розумного чат-боту з технологією NLP. Для досягнення визначеної мети в бакалаврській дипломній роботі пропонується вирішити такі задачі:

- дослідити існуючі методи розробки чат-бота;
- дослідити існуючі проблеми чат-ботів;

- визначення програмної платформи, а саме фреймворків та інтерфейс прикладних рішень і бібліотек API з великою кількістю доступних інструментів: функцій, класів, структур, які підходять для сучасної розробки бота;
- дослідити методи машинного навчання для вдосконалення ботів, та їх взаємодію з користувачем;
- розробка прикладної програми.

Робота виконана згідно з вимогами стандарту ДСТУ 3008-2016 та методичними вказівками щодо розробки та оформлення бакалаврської дипломної роботи за спеціальністю 122 «Комп'ютерні науки» та освітньою програмою «Інформаційні управляючі системи та технології».

РОЗДІЛ 1

ЗАВДАННЯ ОБРОБКИ ПРИРОДНОЇ МОВИ

1.1. Загальний опис обробки природної мови

Обробка природної мови (англ. Natural-language processing, NLP) — загальний напрям інформатики, штучного інтелекту та математичної лінгвістики. Він вивчає проблеми комп'ютерного аналізу та синтезу природної мови. Стосовно штучного інтелекту аналіз означає розуміння мови, а синтез — генерацію розумного тексту. Розв'язок цих проблем буде означати створення зручнішої форми взаємодії комп'ютера та людини.

Як казала дослідниця в області ШІ Елізабет Лідді, обробка природної мови — це комп'ютеризований підхід до аналізу тексту, що базується на низці теорій та наборі технологій. Ця галузь не має одного загальноприйнятого визначення, адже вона перебуває у стані постійних досліджень та розробок. Однак, існують певні аспекти, які б об'єднували усі існуючі визначення.[1]

Можна визначити обробку природної мови як міждисциплінарна галузь науки, що охоплює методики обчислювальної лінгвістики та теорії ШІ, основною метою якої є забезпечення вербальної та невербальної комунікації між людьми та комп'ютерними системами.

1.2. Історія обробки природної мови

Обробка природної мови сягає корінням у 1950-ті роки. Вже в 1950 році Алан Тьюрінг опублікував статтю під назвою "Обчислювальна техніка та інтелект", в якій запропонував те, що зараз називається тестом Тьюрінга, як критерій інтелекту,

Кафедра КІТ				НАУ 21 16 72 000 ПЗ			
Виконав	Марчук В.С.			Завдання обробки природної мови	Літера	Аркуш	Аркушів
Керівник	Зіатдінов Ю.К.					10	17
Консульт.					411 122		
Н-котрол.	Шевченко О.П.						
Зав. каф.	Савченко А.С.						

завдання, яке включає автоматизовану інтерпретацію та генерацію природної мови, але на той час не сформульоване як проблема, окремо від штучного інтелекту.[2]

Передумова виникнення NLP добре узагальнена експериментом Джона Серла з китайською кімнатою: отримавши збірник правил (наприклад, китайський розмовник, із запитаннями та відповідями), комп'ютер імітує розуміння природної мови (або інші завдання NLP), застосовуючи надані правила до даних, з якими стикається.

Експеримент у Джорджтауні в 1954 році передбачав повністю автоматичний переклад понад шестидесяти російських речень англійською мовою. Автори стверджували, що протягом 3-5 років машинний переклад буде вирішеною проблемою.[3] Однак реальний прогрес був набагато повільнішим, і після звіту ALPAC у 1966 р., який показав, що десятирічні дослідження не змогли перевершити сподівання, фінансування машинного перекладу різко зменшилося. Подальші дослідження машинного перекладу проводились лише наприкінці 1980-х років, коли були розроблені перші статистичні системи машинного перекладу.

Деякими особливо успішними системами обробки природних мов, розробленими в 1960-х роках, були SHRDLU , система природних мов, що працює в обмежених "блокових світах" (алгоритм планування доменів в III) із обмеженими словниками, та ELIZA , імітація "рогерійського психотерапевта", написана Джозефом Вайценбаумом між 1964 і 1966 роками.[4] Не використовуючи майже жодної інформації про людські думки чи емоції, ELIZA іноді забезпечувала вражаючу людську взаємодію. Коли "пацієнт" перевищив дуже малу базу знань, ELIZA могла дати загальну відповідь, наприклад, відповівши на "У мене болить голова" таким запитанням "Чому ти кажеш, що болить голова?".

Протягом 1970-х років багато програмістів почали писати "концептуальні онтології", які структурували реальну інформацію в зрозумілі для комп'ютера дані. Прикладами є MARGIE (Schank, 1975), SAM (Cullingford, 1978), PAM (Wilensky, 1978), TaleSpin (Meehan, 1976), QUALM (Lehnert, 1977), Politics (Carbonell, 1979) і Plot Units (Lehnert 1981). У цей час було написано перші чат-боти (наприклад, PARRY).

1980-ті і початок 1990-х років відзначають розквіт природної обробки мови. Основними напрямками були дослідження синтаксичного аналізу на основі правил, морфології, семантики, посилянь та інших областей розуміння природної мови. Інші напрямки досліджень були продовжені, наприклад, розвиток з віртуальним співрозмовником Racter і Jabberwacky.

До 1980-х років більшість систем обробки природних мов базувались на складних наборах рукописних правил. Однак, починаючи з кінця 1980-х, відбулася революція в обробці природної мови із запровадженням алгоритмів машинного навчання для обробки мови. Це було обумовлено постійним збільшенням обчислювальної потужності, так і поступовим зменшенням домінування Хомських теорій лінгвістики (трансформаційна граматика)[5], теоретичні основи яких знеохочували той тип корпусної лінгвістики, який лежить в основі підходу до машинного навчання до мовної обробки.

Багато помітних ранніх успіхів щодо статистичних методів у NLP відбулися в галузі машинного перекладу, особливо завдяки роботі IBM Research. Ці системи змогли скористатися перевагами існуючих багатомовних текстових корпусів, вироблених парламентом Канади та Європейським Союзом як результат законів, що передбачають переклад усіх урядових проваджень на всі офіційні мови відповідних систем управління. Однак більшість інших систем залежали від корпусів, спеціально розроблених для завдань, що реалізуються цими системами, що було (і часто продовжує бути) основним обмеженням успіху цих систем. Як результат, велика кількість досліджень проводилися методами ефективного навчання на обмежених обсягах даних.

Із зростанням Інтернету з середини 1990-х років стає доступним зростаючий обсяг необроблених мовних даних. Таким чином, дослідження все більше зосереджуються на алгоритмах навчання без нагляду та напівконтролю. Такі алгоритми можуть вчитися на даних, які не були анотовані вручну бажаними відповідями або з використанням комбінації анотованих та неанотованих даних. Як правило, це завдання набагато складніше, ніж навчання під контролем, і, як правило,

дає менш точні результати для заданого обсягу вхідних даних. Однак існує величезна кількість анотованих даних (включаючи, серед іншого, весь вміст Всесвітньої павутини), що часто може компенсувати гірші результати, якщо використовуваний алгоритм має досить низьку часову складність, щоб бути практичним.

У 2010-х роках представницьке навчання та методи машинного навчання в стилі глибоких нейронних мереж набули широкого поширення в обробці природних мов, частково завдяки безлічі результатів, які показують, що такі методи можуть досягти найсучасніших результатів у багатьох завданнях з природної мови, наприклад, у моделюванні мови, синтаксичному аналізі, та багатьох інших завданнях. Це стає все більш важливим у медицині та охороні здоров'я, де NLP використовується для аналізу приміток та тексту в електронних медичних картах, які в іншому випадку були б недоступні для вивчення під час вдосконалення медичної допомоги.[6]

1.3. Методи обробки природної мови

У перші дні розвитку багато систем обробки мови розроблялись символічними методами, тобто ручним кодуванням набору правил у поєднанні з пошуком у словнику.

Більш нові системи, засновані на алгоритмах машинного навчання, мають багато переваг перед правилами, виготовленими власноруч:

- процедури навчання, що використовуються під час машинного навчання, автоматично фокусуються на найпоширеніших випадках, тоді як при написанні правил від руки часто зовсім не очевидно, куди слід спрямовувати зусилля;
- процедури автоматичного навчання можуть використовувати алгоритми статистичного висновку для створення моделей, стійких до незнайомого введення (наприклад, вміст слів або структури, яких раніше не бачили), і до помилкового введення (наприклад, з неправильно написаними словами або словами, випадково пропущеними). Як правило, витончено обробляти такі введення за допомогою

рукописних правил або, загальніше, створювати системи рукописних правил надзвичайно складно і займає багато часу;

- системи, засновані на автоматичному вивченні правил, можуть бути більш точними, просто надаючи більше вхідних даних. Однак системи, засновані на рукописних правилах, можна зробити точнішими лише за рахунок збільшення складності правил, що є набагато складнішим завданням. Зокрема, існує обмеження складності систем, заснованих на рукописних правилах, поза якими системи стають дедалі більше некерованими. Однак створення більшої кількості даних для введення в системи машинного навчання просто вимагає відповідного збільшення кількості відпрацьованих людських годин, як правило, без значного збільшення складності процесу анотування.

Незважаючи на популярність машинного навчання в дослідженнях NLP, символічні методи все ще використовуються:

- коли обсяг навчальних даних є недостатнім для успішного застосування методів машинного навчання, наприклад, для машинного перекладу мов з низьким рівнем ресурсів, таких як передбачені системою Apertium;
- для попередньої обробки, наприклад, токенізації;
- для подальшої обробки та перетворення вихідних даних конвеєрів NLP, наприклад, для вилучення знань із синтаксичних розборів.[7]

Наприкінці 1980-х - середині 1990-х років багато досліджень обробки природних мов значною мірою спиралися на машинне навчання. Натомість парадигма машинного навчання вимагає використання статистичного висновку для автоматичного засвоєння таких правил шляхом аналізу великих корпусів (множинна форма корпусу - це набір документів, можливо, із людськими або комп'ютерними анотаціями) типових реальних прикладів.[8]

Багато різних класів алгоритмів машинного навчання було застосовано до завдань з обробки природної мови. Ці алгоритми беруть за вхідні дані великий набір "функцій", що генеруються з вхідних даних. Однак все частіше дослідження зосереджуються на статистичних моделях, які приймають м'які, ймовірнісні рішення,

засновані на додаванні реальних вагових коефіцієнтів до кожної вхідної ознаки. Такі моделі мають ту перевагу, що вони можуть виражати відносну визначеність багатьох різних можливих відповідей, а не лише одну, даючи більш надійні результати, коли така модель включається як компонент більшої системи.

Деякі з найбільш ранніх алгоритмів машинного навчання, такі як дерево рішень, створили системи жорстких правил if-then, подібних до існуючих рукописних правил. Однак позначення частковою мовою познайомило використання прихованих марковських моделей з обробкою природних мов, і все частіше дослідження зосереджувались на статистичних моделях, які приймають м'які, ймовірнісні рішення, засновані на додаванні реальних значень ваги до ознак, що складають вхідні дані. Кешовані модельні мови, на які сьогодні покладаються, є прикладами таких статистичних моделей. Такі моделі, як правило, є більш надійними, якщо їм надається незнайоме введення, особливо введення, яке містить помилки (як це дуже часто зустрічається у реальних даних), і дають більш надійні результати, якщо їх інтегрують у більшу систему, що містить безліч підзадач.

Починаючи з «нейронного повороту», статистичні методи в дослідженнях NLP в основному замінялись нейронними мережами. Однак вони продовжують бути актуальними для контекстів, в яких необхідна статистична інтерпретація та прозорість.

Основним недоліком статистичних методів є те, що вони вимагають детальної інженерії характеристик. Починаючи з 2015 року, ця сфера в основному відмовилася від статистичних методів і перейшла на нейронні мережі для машинного навчання. Популярні прийоми включають використання вбудованих семантичних властивостей слів та збільшення наскрізного вивчення завдання вищого рівня (наприклад, відповіді на запитання) замість того, щоб покладатися на конвеєр окремих проміжних завдань (наприклад, позначення частини мови та залежність від синтаксичного розбіру). У деяких областях цей зсув спричинив суттєві зміни в тому, як розробляються системи NLP, такі, що глибокі нейромережеві підходи можуть розглядатися як нова парадигма, відмінна від статистичної обробки природних мов. Наприклад, термін

нейронний машинний переклад (NMT) підкреслює той факт, що підходи до машинного перекладу, засновані на глибокому навчанні, безпосередньо вивчають перетворення послідовності в послідовність, позбавляючи потреби проміжних етапів, таких як вирівнювання слів та моделювання мови, що використовувався в статистичному машинному перекладі (SMT).

1.4. Поширені завдання обробки природної мови

Далі наведено перелік найбільш часто досліджуваних завдань в обробці природної мови. Деякі з цих завдань мають прямі реальні програми, тоді як інші частіше служать підзавданнями, які використовуються для вирішення великих завдань.[9]

Незважаючи на те, що завдання з обробки природної мови тісно взаємопов'язані, для зручності їх можна розділити на категорії:

1) обробка тексту та мови:

- оптичне розпізнавання тексту (отримавши зображення, що представляє собою друкований текст, визначається відповідний текст);
- розпізнавання мовлення (у більшості розмов звуки, що представляють послідовні літери, змішуються один з одним у процесі, який називається коартикуляцією, тому перетворення аналогового сигналу у дискретні символи може бути дуже складним процесом. Крім того, враховуючи те, що слова люди вимовляють з різними наголосами, програмне забезпечення розпізнавання мови повинно мати можливість розпізнавати широкий спектр введених даних як ідентичні один одному з точки зору його текстового еквівалента);
- сегментація мовлення (розділення мовлення на слова, підзадача розпізнавання мовлення);
- синтез мовлення (перетворення тексту в усне подання);

- сегментація слів, або токенізація (розкладання суцільного тексту на окремі слова. Для такої мови, як англійська, це досить тривіально, оскільки слова зазвичай розділяються пробілами. Однак деякі письмові мови, такі як китайська, японська та тайська, не позначають межі слів таким чином, і в цих мовах сегментація тексту є важливим завданням, що вимагає знання словникового запасу та морфології слів у мові) (рис. 1.1);



Рис. 1.1. Результат роботи простого токенізатора

2) морфологічний аналіз:

- лемматизація (методика зведення слів до їх нормованої форми. Трансформація фактично використовує словник для відображення слів у їх фактичній формі) (рис. 1.2) ;

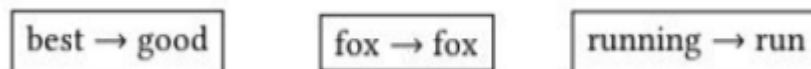


Рис. 1.2. Приклад роботи лематизації

- морфологічна сегментація (поділення слів на окремі морфеми та визначення класів морфем. Складність цього завдання в значній мірі залежить від складності морфології мови, що розглядається. В індійській мові такий підхід неможливий, оскільки кожен словниковий запис містить тисячі можливих форм слів);

- розмічування частин мови (вказавши речення, визначення частини мови для кожного слова. Багато слів, особливо загальноживані, можуть служити кількома частинами мови);
- стемінг (процес зменшення складених (або іноді похідних) слів до базової форми. Дає результати, подібні до лематизації, але робить це на основі правил, а не словника) (рис. 1.3);

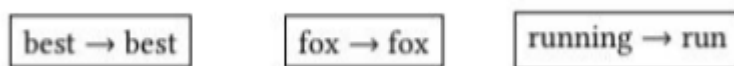


Рис. 1.3. Приклад стеммінгу

3) синтаксичний аналіз:

- індукція граматики (створення офіційної граматики, яка описує синтаксис мови);
- порушення речення (за допомогою шматків тексту, знаходження межі речень. Межі речень часто позначаються крапками або іншими розділовими знаками, але ці самі символи можуть служити й іншим цілям (наприклад, позначення скорочень));
- синтаксичний аналіз (визначення граматичного аналізу речень. Граматика для природних мов є неоднозначною і типові пропозиції мають кілька можливих аналізів);

4) лексична семантика (окремих слів у контексті):

- лексична семантика (обчислення значень окремих слів у контексті);
- дистрибутивна семантика (засвоєння семантичних подань з даних);
- розпізнавання іменованих сутностей (враховуючи потік тексту, визначення елементів в тексті які відображаються на власні імена);
- аналіз тональності тексту (витягування суб'єктивної інформації, як правило, з набору документів, часто використовуючи огляди в Інтернеті для визначення "полярності" щодо конкретних об'єктів);

- вилучення термінології (автоматичне вилучення відповідних термінів із даного корпусу);
- дозвіл лексичної багатозначності (багато слів мають більше одного значення, потрібно вибрати значення, яке має найбільший сенс у контексті);

5) реляційна семантика (семантика окремих речень):

- витяг відносин (з огляду на шматок тексту, визначення взаємозв'язків між названими сутностями);
- семантичний синтаксичний розбір (враховуючи фрагмент тексту, подання його офіційного представлення семантики або як графік, або відповідно до логічного формалізму. Цей виклик, як правило, включає аспекти декількох елементарних завдань NLP із семантики (наприклад, семантичне маркування ролей, неоднозначність сенсу слова) і може бути розширений, включаючи повноцінний аналіз дискурсу);
- семантичне маркування ролей (враховуючи одне речення, визначення та роз'єднання семантичних предикатів (наприклад, словесні фрейми), а потім ідентифікація та класифікація фреймів);

б) дискурс (семантика поза окремими реченнями):

- кореферентність (враховуючи речення або більший шматок тексту, визначення, які слова стосуються сутностей. Роздільна здатність анафори є конкретним прикладом цього завдання, і саме вона стосується зіставлення займенників з іменниками чи іменами, до яких вони відносяться);
- дискурсивний аналіз (виявлення дискурсної структури зв'язаного тексту, тобто природи дискурсивних зв'язків між реченнями. Іншим можливим завданням є розпізнавання та класифікація мовленнєвих актів у частині тексту (наприклад, запитання так-ні, запитання про зміст, висловлювання, твердження тощо);

- неявне семантичне маркування ролей (враховуючи одне речення, визначення та роз'єднання семантичних предикатів та їх явних семантичних ролей в поточному реченні);
- визнання текстового втягування (враховуючи два фрагменти тексту, визначення, чи один з них відповідає дійсності, чи заперечує іншому);
- сегментація та розпізнавання тем (з огляду на шматок тексту, розділення його на сегменти, кожен з яких присвячений темі, та визначення теми сегмента);
- видобуток аргументів (автоматичне вилучення та ідентифікація аргументованих структур із тексту природною мовою за допомогою комп'ютерних програм);

7) програми NLP вищого рівня:

- автоматизоване реферування (створення читабельного висновку фрагмента тексту. Часто використовується для подання коротких викладів тексту знайомого типу, таких як наукові праці, статті);
- створення книг (перша наукова книга, створена машиною, була опублікована в 2019 році (Beta Writer, Lithium-Ion Battery , Springer, Cham), ґрунтується на фактичних знаннях і базується на узагальненні тексту);
- документ AI (дозволяє користувачам, які не мали досвіду роботи з штучним інтелектом, машинним навчанням або NLP, швидко навчити комп'ютер витягувати конкретні дані, які їм потрібні, з різних типів документів);
- граматичне виправлення помилок (виявлення та виправлення граматичних помилок передбачає велику кількість проблем на всіх рівнях лінгвістичного аналізу (фонологія / орфографія, морфологія, синтаксис, семантика, прагматика));

- машинний переклад (автоматичне перекладання тексту з однієї людської мови на іншу. Це одна з найскладніших проблем, і вона входить до класу проблем, що в просторіччі називається " AI-complete ", тобто вимагає всіх різних типів знань, якими володіє людина (граматика, семантика, факти про реальний світ тощо));
- генерація природної мови (NLG) (перетворення інформації з комп'ютерних баз даних на читабельну людську мову);
- розуміння природної мови (NLU) (перетворення фрагментів тексту на більш офіційні подання, такі як логічні структури першого порядку, якими легше керувати комп'ютерним програмам. Розуміння природної мови передбачає ідентифікацію передбачуваної семантики з безлічі можливих семантик, яка може бути виведена із виразу природної мови, який зазвичай має форму організованих позначень понять природної мови);
- питально-відповідна система (враховуючи людське запитання, визначення відповіді на нього).

1.5. Трансформери в обробці природної мови

Трансформер (англ. Transformer) — це модель глибокого навчання, що була представлена у 2017 році, використовується переважно в області обробки природної мови.

Як і рекурентні нейронні мережі (РНМ), трансформери використовуються для обробки послідовних даних, таких як природна мова, для таких задач як переклад тексту. Проте, на відміну від РНМ, трансформери не вимагають, щоби ці послідовні дані оброблювали за порядком. Наприклад, якщо вхідні дані є реченням, то трансформеру не потрібно обробляти його початок, перш ніж взятися за обробку кінця речення. Тому трансформер уможливорює більше розпаралелювання, ніж РНМ,

і тому зменшується тривалість тренування.[10] На рисунку 1.4 показано як рекурентна нейронна мережа прогнозує результат.

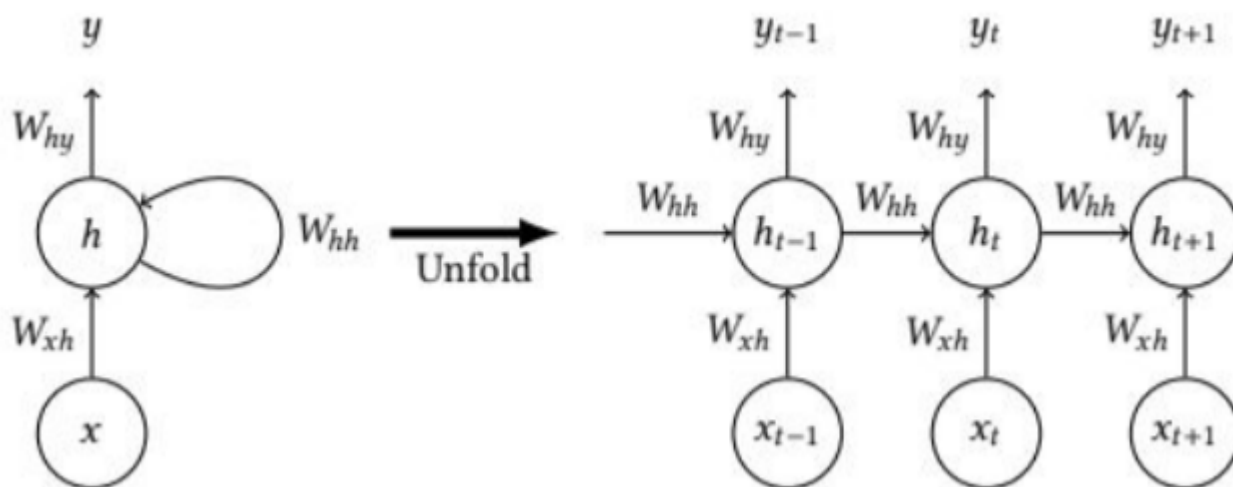


Рис. 1.4. Рекурентна нейронна мережа прогнозує результат

Трансформери стали популярними моделями для вирішення багатьох задач в NLP, замінивши старіші моделі рекурентних нейронних мереж. Оскільки модель трансформер сприяє більшому розпаралелюванню під час тренування, вона уможливила тренування на більших наборах даних, ніж це було можливо до її представлення. Це призвело до розробки попередньо натренованих систем, таких як BERT (англ. Bidirectional Encoder Representations from Transformers) та GPT (англ. Generative Pre-trained Transformer), які було натреновано на величезних універсальних мовних наборах даних, і які можливо тонко налаштовувати під конкретні мовні задачі.[11]

Трансформер має кодувальню-декодувальню архітектуру. Кодувальник складається з набору кодувальних шарів, що оброблюють вхід ітеративно, шар за шаром, а декодувальник складається з набору декодувальних шарів, які роблять те саме з виходом кодувальника.

Функцією кожного з кодувальних шарів є обробка своїх вхідних даних, щоби породжувати кодування, що містить інформацію про те, які вхідні частини є

релевантними одна до одної. Як кодувальні, так і декодувальні шари мають нейронну мережу прямого поширення для додаткової обробки виходів, та містять залишкові з'єднання й кроки унормовування шару.[12]

Основою трансформера є вузли масштабованої скалярно-добуткової уваги (англ. scaled dot-product attention units). Коли трансформеру передають речення, ваги уваги між усіма лексемами обчислюються одночасно. Вузол уваги виробляє вкладання для кожної лексеми в контексті, який містить інформацію не лише про саму лексему, але й про зважене поєднання інших релевантних лексем, зважених за вагами уваги.

Один набір матриць називають головою уваги (англ. attention head), й кожен шар у моделі трансформера має декілька голів уваги. В той час як одна голова уваги звертає увагу на лексеми, що є релевантними до кожної з лексем, з декількома головами уваги ця модель може навчатися робити це для різних визначень «релевантності». Дослідники показали, що багато голів уваги у трансформерах кодують відношення релевантності, що є інтерпретованими для людей. Наприклад, є голови уваги, що для кожної лексеми звертають увагу переважно на наступне слово, або голови уваги, що переважно звертають увагу дієслів на їхні безпосередні об'єкти. Оскільки моделі трансформера мають декілька голів уваги, вони мають можливість схоплювати багато рівнів та типів відношень релевантності, від поверхневих до семантичних. Декілька виходів шару багатоголової уваги зчіплюють, щоби передавати їх до шарів нейронних мереж прямого поширення.[13]

Кожен кодувальник (англ. encoder) складається з двох головних складових: механізму самоуваги (англ. self-attention mechanism), та нейронної мережі прямого поширення. Механізм самоуваги бере кодування входу з попереднього кодувальника, та зважає їхню релевантність одне одному, щоби породити набір кодувань виходу. Нейронна мережа прямого поширення відтак здійснює подальшу обробку кожного кодування виходу окремо. Зрештою, ці кодування виходу передаються наступному кодувальникові як його входу, так само як і декодувальникам.[10]

Кожен декодувальник (англ. decoder) складається з трьох головних складових: механізму самоуваги, механізму уваги над кодуваннями, та нейронної мережі прямого поширення. Декодувальник працює подібно до кодувальника, крім вставленого додаткового механізму уваги, що натомість дістає релевантну інформацію з кодувань, породжених кодувальниками.[10]

Трансформери, як правило, підлягають напівавтоматичному навчанню, що включає спонтанне попереднє тренування, з наступним керованим тонким налаштуванням. Попереднє тренування, як правило, виконують на набагато більшому наборі даних, ніж тонке налаштування, через обмежену доступність мічених тренувальних даних. До задач попереднього тренування та тонкого налаштування зазвичай належать:

- передбачення наступного речення;
- відповідання на питання;
- розуміння прочитаного тексту;
- аналіз тональності тексту;
- автоматизоване реферування.

Трансформер застосовується в області обробки природної мови, наприклад, в задачах машинного перекладу. Багато попередньо натренованих моделей, таких як GPT-3, GPT-2, BERT, XLNet та RoBERTa, демонструють здатність трансформерів виконувати широкий спектр пов'язаних з NLP задач. У 2020 році було продемонстровано, що модель GPT-2 може грати у шахи.[14]

1.6. Чат-бот як реалізація моделі обробки природної мови

Чатбот — це програма штучного інтелекту, яка імітує інтерактивну розмову людини за допомогою ключових, заздалегідь розрахованих фраз користувача, та слухових або текстових сигналів.[15] Термін «чатер-бот» (англ. ChatterBot) вперше вжив Майкл Маулдін у 1994 році для опису розмовних програм, які дозволяють спілкування між людиною і чат-ботом, який наділений штучним інтелектом. Чат-боти

є частиною віртуальних помічників, таких як Google Assistant, і доступні через програми багатьох організацій, веб-сайти та платформи обміну миттєвими повідомленнями.

У чат-боті отримання інформації або виконання інших задач відбувається шляхом вибору запропонованих відповідей, або запропонованих категорій. Більш складні боти надають можливість вводу тексту і отримання інформації згідно до запиту користувача.

Першими чат-ботами в історії вважаються ELIZA (1966) і PARRY (1972). Далі були створені: A.L.I.C.E., Jabberwacky, D.U.D.E. та ін.

Чат-бот PARRY[16] створив американський психіатр Кеннетом Колбі в 1972 році. Чат-бот імітував реакції пацієнта, хворого на параноїдальну шизофренію, що нагадувало мислення звичайної людини. PARRY використовував складну систему припущень, атрибуцій та «емоційних відповідей», його роботу на початку 1970-х років перевіряли через тест Тюрінга. Чат-бот Jabberwacky дізнається нові відповіді та контекст, що базуються на взаємодії користувачів в режимі реального часу, і не керується статичною базою даних. На сьогодні немає розмовного штучного інтелекту загального призначення, і деякі розробники програмного забезпечення зосереджують увагу на практичному аспекті пошуку інформації.

За сферою застосування чат-боти поділяються на персональні (для особистого спілкування) та споживчі (підтримка клієнтів компанії на корпоративному сайті та в мобільних додатках).

Висновки до розділу 1

У даному розділі була сформована задача даної роботи, а також були розглянуті базові принципи обробки природної мови (Natural language processing), та сучасні методи використання машинного навчання та нейронних мереж для обробки природної мови. Аналіз даних з повідомлень в соцмережах є одним з найбільш яскравих прикладів використання методів NLP, оскільки можна працювати з великими масивами даних в реальному часі, та приносити користь соціуму за допомогою фільтрації небажаного контенту, рекомендаційних сіток. Однак це є не єдиним напрямом використання NLP. NLP широко використовується в голосових помічниках, автовідповідачах, оцифруванні мови, створенні субтитрів, знаходженні взаємозв'язків у текстах, виведенні головної ідеї з тексту, аналізі дискурсу, прогнозуванні та підказках при друці, виправленні граматичних, синтаксичних, лексичних помилок.

РОЗДІЛ 2

ТЕХНОЛОГІЇ ТА ЗАСОБИ ОБРОБКИ ПРИРОДНОЇ МОВИ

2.1. Огляд мовних моделей для обробки природної мови

Впровадження передавального навчання та навчених мовних моделей у NLP розсунуло межі розуміння та генерації мови, і наразі це дуже популярні методи при обробці природної мови.

Але нині існує суперечка щодо дослідницької цінності величезних попередньо навчених мовних моделей, що займають таблиці лідерів. Хоча багато експертів зі штучного інтелекту погоджуються із твердженням Анни Роджерс про те, що отримання сучасних результатів просто за допомогою більшої кількості даних та обчислювальних потужностей не є новиною для досліджень, інші лідери думок щодо NLP вказують на деякі позитивні моменти в поточній тенденції, наприклад, в можливості побачити основні обмеження поточної парадигми.

У будь-якому випадку, останні вдосконалення мовних моделей NLP, схоже, зумовлені не тільки значним збільшенням обчислювальних можливостей, але також відкриттям винахідливих способів полегшення моделей при збереженні високої продуктивності.

Щоб розібратися в останніх проривах у мовному моделюванні, проведемо аналіз популярних мовних моделей, впроваджених протягом останніх кількох років.

2.1.1. BERT: тренування глибоких двонаправлених трансформерів

BERT (англ. Bidirectional Encoder Representations from Transformers, двоспрямовані кодувальні представлення з Трансформерів) — модель, призначена для попередньої підготовки глибоких двоспрямованих представлень шляхом

Кафедра КІТ				НАУ 21 16 72 000 ПЗ			
Виконав	Марчук В.С.			Технології та засоби обробки природної мови	Літера	Аркуш	Аркушів
Керівник	Зіатдінов Ю.К.					27	26
Консульт.					411 122		
Н-котрол.	Шевченко О.П.						
Зав. каф.	Савченко А.С.						

спільного кондиціонування як лівого, так і правого контексту у всіх сторонах. Як результат, заздалегідь навчені представлення BERT можуть бути відрегульовані лише за допомогою одного додаткового рівня виведення, щоб створити найсучасніші моделі для широкого кола завдань, таких як відповідь на запитання та мовний висновок, без істотних модифікацій архітектури, специфічних для конкретного завдання.[17]

BERT концептуально проста та емпірично потужна. Вона отримує нові найсучасніші результати по одинадцяти завданням NLP, включаючи досягнення бенчмарку GLUE до 80,4% (абсолютне покращення на 7,6%), точності MultiNLI до 86,7 % (абсолютне покращення на 5,6%) та проходження тесту F1 SQuAD v1.1 на 93,2 % (абсолютне покращення на 1,5%), перевершуючи результати людини на 2,0%.

Команда Google AI зробила ультрасучасну модель для обробки природної мови. Її конструкція дозволяє моделі розглядати контекст як з лівої, так і з правої сторони кожного слова. Будучи концептуально простою, BERT отримує нові найсучасніші результати в одинадцяти завданнях NLP, включаючи відповіді на запитання, розпізнавання іменних сутностей та інші завдання, пов'язані із загальним розумінням мови. На рисунку 2.1 показано порівняння архітектур розглядаємої BERT та OpenAI GPT, що буде розглянута далі.

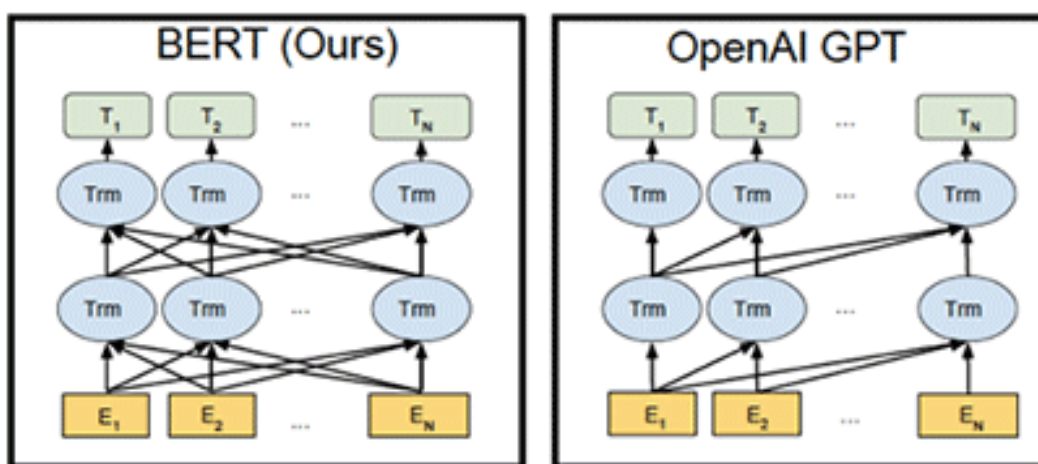


Рис. 2.1. Порівняння архітектур моделей BERT та OpenAI GPT

Сенс моделі BERT:

- навчання шляхом випадкового маскування відсотку вхідних токенів, уникаючи циклів, коли слова можуть опосередковано «бачити себе»;
- навчання через модель взаємозв'язку речень, побудувавши просте завдання бінарної класифікації, щоб передбачити, чи речення B безпосередньо слідує за реченням A, що дозволяє BERT краще зрозуміти взаємозв'язки між реченнями;
- тренування дуже великої моделі (24 блоки трансформерів, 1024 прихованих, 340М параметрів) з великою кількістю даних (3,3 мільярда наборів слів).

BERT може надавати допомогу підприємствам із широким колом проблем з NLP, включаючи: чат-боти для кращої взаємодії з клієнтами; аналіз відгуків клієнтів; пошук відповідної інформації тощо.

2.1.2. GPT-2: багатозадачна неконтрольована модель

GPT-2 – модель-трансформер, яка досягає найкращих результатів в 7 з 8 перевірених наборів даних моделювання мови, і підходить для WebText. WebText – метод контрольованого тренування моделей без явного нагляду на даних з мільйонів веб-сторінок. Це вказує на перспективний шлях до побудови систем обробки мови, які вчать виконувати завдання на основі своїх природних демонстрацій. Команда OpenAI демонструє, що заздалегідь навчені мовні моделі можуть використовуватися для вирішення подальших завдань без будь-яких змін параметрів або архітектури. Вони навчили дуже велику модель, трансформер із параметрами 1,5B (півтора мільярда), на великому та різноманітному наборі даних, що містить текст, викреслений з 45 мільйонів веб-сторінок (рис. 2.2). Модель формує послідовні абзаци тексту та досягає перспективних, конкурентних і найкращих результатів у найрізноманітніших завданнях.[19]

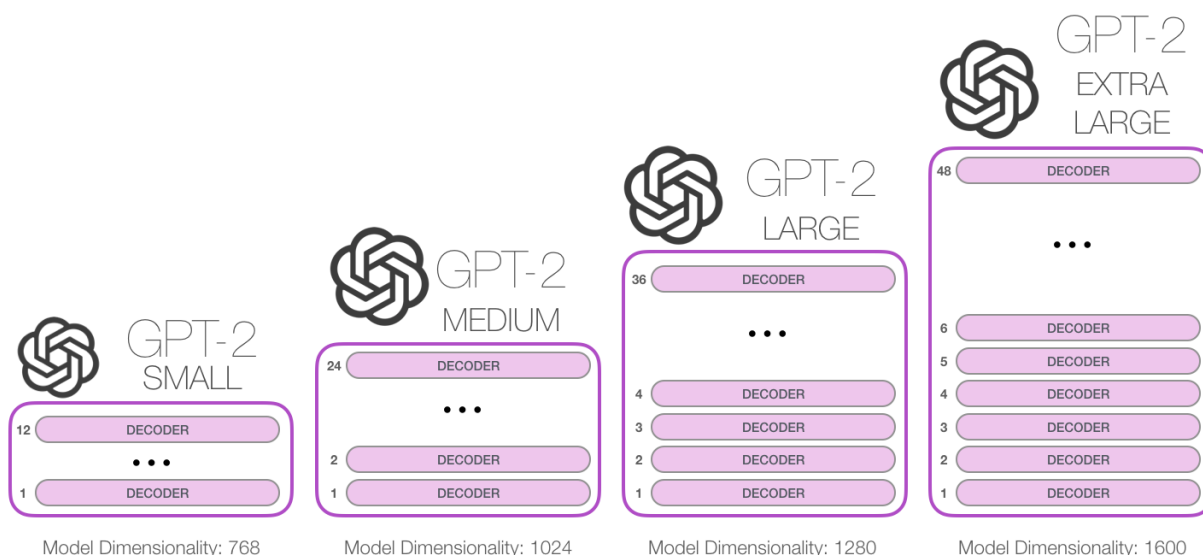


Рис. 2.2. Порівняння розмірів попередньо навчених моделей GPT-2

Навчається дана модель на великому та різноманітному наборі даних:

- веб-сторінки, які фільтруються людьми;
- очищених текстах та з видаленням всіх документів Вікіпедії для мінімізації накладання навчальних та тестових наборів;
- з використанням отриманого набору даних WebText із трохи більше 8 мільйонами документів на загальну кількість 40 ГБ тексту.

Модель використовує версії байтового кодування пари байтів (BPE) для представлення вхідних даних. Нині найбільша модель включає 1542 мільйонів параметрів і 48 шарів; модель переважно відповідає моделі OpenAI GPT із невеликими змінами (тобто, розширенням словникового запасу та розміром контексту, зміною ініціалізації тощо).

GPT2 демонструє досить перспективні результати у здоровому глузді міркувань, відповідей на запитання, розуміння читання та перекладу. Вона створює зв'язні тексти, наприклад, статті новин про відкриття розмовляючих єдинорогів.

В цій моделі потрібно досліджувати тонке налаштування таких бенчмарків, як decaNLP та GLUE, щоб з'ясувати, чи можуть величезні набори даних та потужність GPT-2 подолати неефективність односпрямованих представлень BERT.

Що стосується практичних застосувань, модель GPT-2 без точного налаштування далеко не придатна для використання, але вона показує дуже перспективний напрямок досліджень.

2.1.3. XLNet: узагальнена авторегресивна попередня підготовка

Завдяки можливості моделювання двонаправлених контекстів, попередження підготовки на основі автоматичного кодування, як BERT, досягає кращих показників, ніж підходи до попередньої підготовки на основі авторегресивного моделювання мови. Однак, покладаючись на псування вхідних даних масками, BERT нехтує залежністю між маскованими позиціями. Існує альтернатива — модель XLNet. Дослідники з університету Карнегі Меллона та Google розробили модель XLNet для завдань з обробки природної мови, таких як розуміння читання, класифікація тексту, аналіз сентиментальності та інші. XLNet - це узагальнений метод авторегресивного преднавчання, який використовує найкращі результати як авторегресивного моделювання мови (наприклад, Transformer-XL), так і автокодування (наприклад, BERT), уникаючи їх обмежень. Експерименти демонструють, що ця модель перевершує BERT і Transformer-XL і досягає найкращих результатів у 18 завданнях NLP.[19]

Як і BERT, XLNet використовує двонаправлений контекст, що означає, що вона розглядає слова перед і після даного токена, щоб передбачити, яким він повинен бути (рис. 2.3). З цією метою XLNet максимізує очікувану вірогідність журналу послідовності щодо всіх можливих перестановок порядку факторизації. Як авторегресивна мовна модель, XLNet не покладається на пошкодження даних, а отже, уникає обмежень BERT через маскування. Для подальшого вдосконалення архітектурних конструкцій для попередньої підготовки XLNet інтегрує механізм повторення сегментів та відносну схему кодування Transformer-XL.

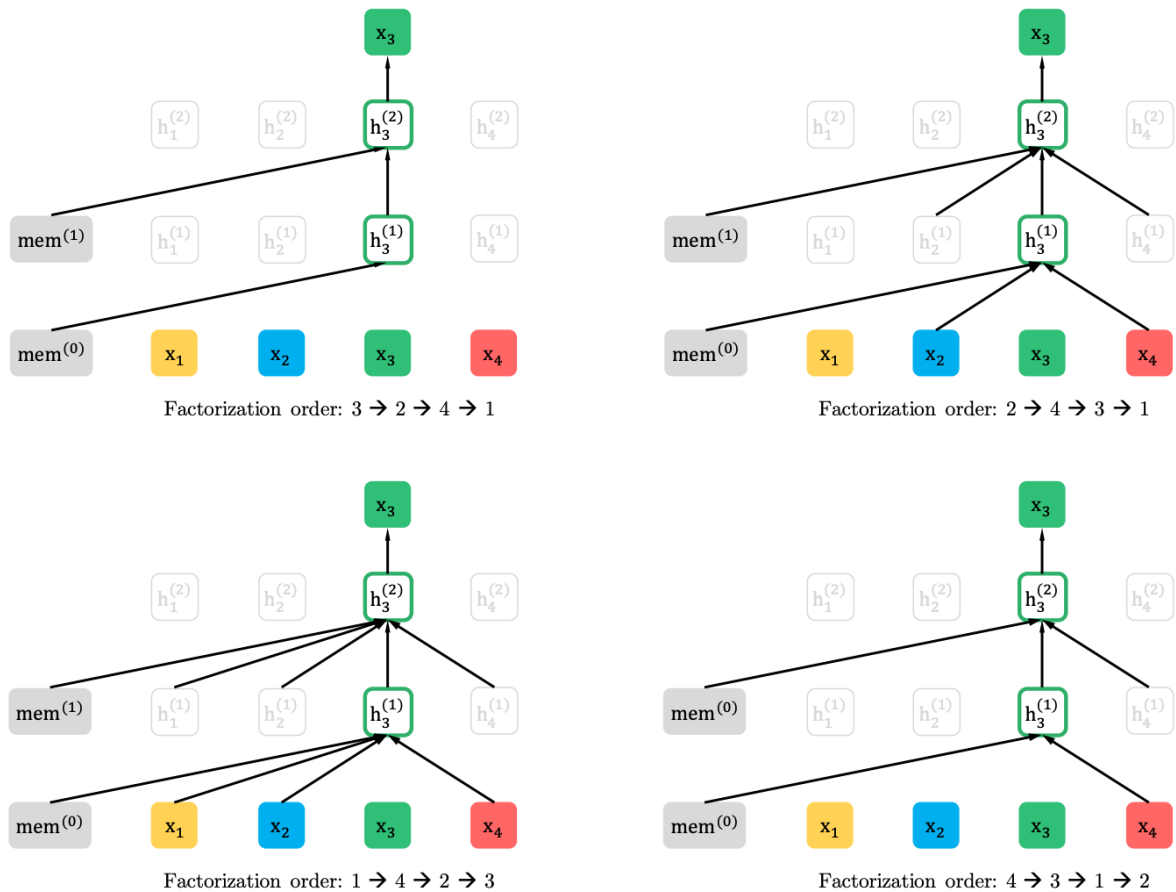


Рис. 2.3. Передбачення токєну x_3
з рїзних перестановок вихїдної послїдовностї

XLNet може надавати допомогу компанїям з широким спектром проблем NLP, зокрема:

- чат-боти для пїдтримки клїєнтїв першої лїнїї або вїдповїдї на запити щодо продуктїв;
- аналїз сентиментальностї для оцїнки проїнформованостї та сприйняття бренду на основї вїдгукїв клїєнтїв та соцїальних мереж;
- пошук вїдповїдної їнформацїї в базах документїв або в Їнтернетї тощо.

2.1.4. RoBERTa: надійно оптимізований підхід до підготовки BERT

Моделі обробки природної мови досягли значного прогресу завдяки впровадженню методів попередньої підготовки, але обчислювальні витрати на навчання ускладнили параметри реплікації та точного налаштування. Facebook AI та дослідники університету Вашингтона проаналізували навчання моделі двостороннього кодування Google від трансформерів (BERT) та виявили кілька змін у навчальній процедурі, що покращують її ефективність. Зокрема, дослідники використали новий, більший набір даних для навчання, навчили модель набагато більшим ітераціям та видалили наступну послідовність прогнозування навчальної мети. Отримана в результаті оптимізована модель RoBERTa (надійно оптимізований підхід BERT) відповідає оцінкам нещодавно представленої моделі XLNet в бенчмарку GLUE.[20] На рисунку 2.4 можна розглянути структуру даної моделі.

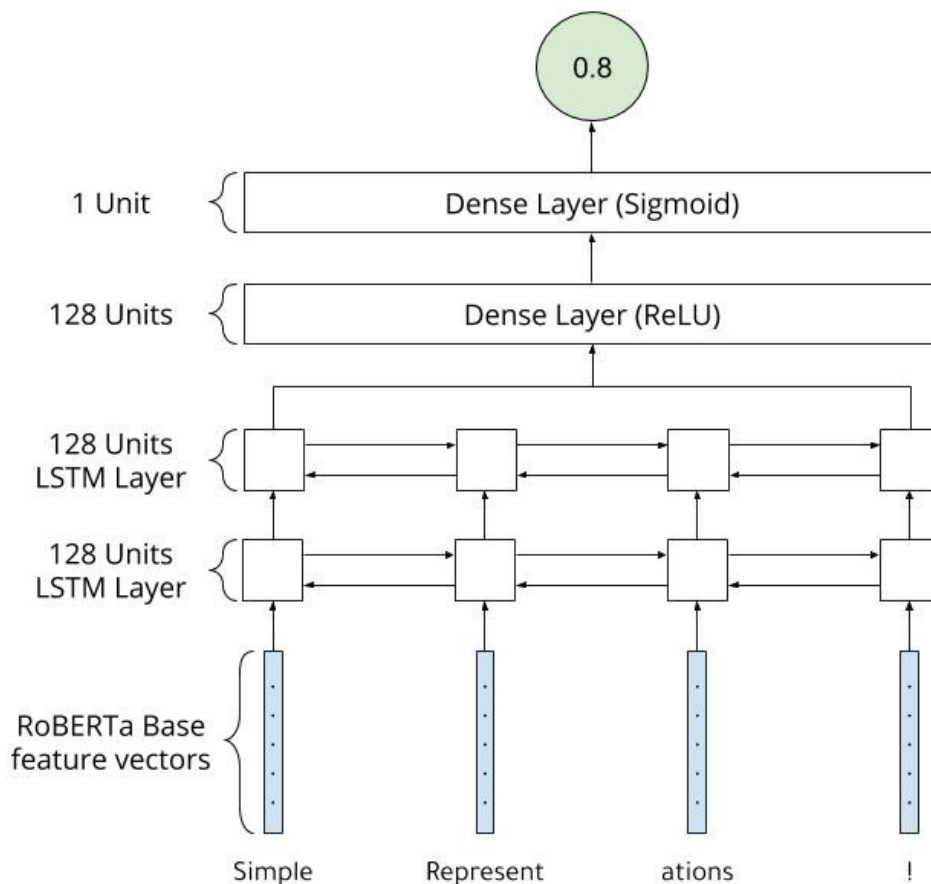


Рис. 2.4. Структура моделі RoBERTa

Дослідницька група Facebook AI виявила, що BERT значно недонавчена, і запропонувала вдосконалений рецепт її навчання, який називається RoBERTa:

- більше даних: 160 Гб тексту замість 16 Гб набору даних, який спочатку використовувався для навчання BERT;
- більш тривале навчання: збільшення кількості ітерацій зі 100 до 300 тисяч, а потім ще до 500 тисяч;
- більші партії: 8 тисяч замість 256 в оригінальній базовій моделі BERT;
- більший словниковий запас VPE на рівні байтів із 50 000 одиниць підслів замість словникового запасу VPE розміром 30 000;
- видалення наступної цілі прогнозування послідовності з навчальної процедури;
- динамічно мінливий шаблон маскування, застосований до даних тренувань.

RoBERTa може удосконалюватися, включаючи більш складні процедури багаторівневого планування. Може використовуватися в бізнес-середовищі для широкого кола подальших завдань, включаючи системи діалогу, відповіді на запитання, класифікацію документів тощо.

2.1.5. ALBERT: легка BERT для самостійного вивчення мовних уявлень

Команда Google Research займається проблемою постійно зростаючого розміру попередньо навчених мовних моделей, що призводить до обмеження пам'яті, збільшення часу навчання та інколи несподівано погіршення продуктивності. Зокрема, вони ввели архітектуру Lite BERT (ALBERT). Дана модель включає два методи зменшення параметрів:[21]

- факторизована параметризація вбудовування, де розмір прихованих шарів відокремлюється від розміру вкладених словникових запасів шляхом розкладання великої матриці, що вкладає словниковий запас, на дві малі матриці;
- спільне використання параметрів між шарами, щоб запобігти зростанню кількості параметрів із глибиною мережі.

Ефективність роботи ALBERT додатково покращується шляхом введення самоконтрольованої втрати для прогнозування порядку речень для усунення обмежень BERT щодо узгодженості між реченнями.

Завдяки впровадженим методам зменшення параметрів, конфігурація ALBERT з $18\times$ меншою кількістю параметрів та $1,7\times$ швидшим навчанням порівняно з оригінальною великою моделлю BERT досягає лише трохи гіршої продуктивності. Значно більша конфігурація ALBERT, яка все ще має менше параметрів, ніж велика BERT, перевершує всі поточні сучасні мовні режими, отримуючи:

- 89,4% точності за тестом RACE;
- 89,4 бали за тестом GLUE;
- F1 92,2% за тестом SQuAD 2.0.

ALBERT може в подальшому покращити свою продуктивність за допомогою важких прикладів майнінгу, ефективнішого навчання моделі та інших підходів. Ця модель може бути використана в бізнес-налаштуваннях для підвищення продуктивності широкого кола подальших завдань, включаючи продуктивність чат-ботів, аналіз сентиментальності, аналіз документів та класифікацію тексту.

2.1.6. StructBERT: включення мовних структур у попередню підготовку

Дослідницька група Alibaba запропонувала розширити BERT до нової мовної моделі StructBERT. Експерименти демонструють, що введена модель суттєво покращує сучасні результати з різних завдань із розуміння природної мови, включаючи аналіз сентиментальності та відповіді на запитання. Модель базується на архітектурі BERT з багат шаровою двонаправленою мережею трансформерів:[22]

- запропонована модель змішує певну кількість токенів після маскуванню слів та прогнозує правильний порядок;
- модель випадковим чином змішує порядок речень і передбачає наступне та попереднє речення як нове завдання передбачення речення;

StructBERT від Alibaba досягає найкращих результатів у різних завданнях NLP;

- за тестом GLUE вона перевершила усі опубліковані моделі за середнім балом і досягла найкращих результатів у 6 із 9 завдань;
- що стосується набору даних SNLI, StructBERT перевершила усі існуючі підходи, отримавши найкращий результат - 91,7%;
- щодо тесту відповідей на запитання SQuAD 1.1, ця модель перевершила всі опубліковані моделі, за винятком XLNet із збільшеними даними.

Як і інші попередньо навчені мовні моделі, StructBERT може допомагати компаніям у виконанні різноманітних завдань NLP, включаючи відповіді на запитання, аналіз сентиментальності, узагальнення документів тощо.

2.1.7. T5: вивчення меж передавального навчання

Дослідницька група Google запропонувала уніфікований підхід до передачі навчання в NLP з метою встановлення нового рівня технології в цій галузі. З цією метою вони запропонували розглядати кожну проблему NLP як проблему "перетворення тексту в текст". Така структура дозволяє використовувати одну і ту ж модель, ціль, процедуру навчання та процес декодування для різних завдань, включаючи узагальнення, аналіз сентиментальності, відповіді на запитання та машинний переклад. Дослідники називають свою модель Text-to-Text Transfer Transformer (T5)[23] і навчають її на великому зборі даних, зібраних з Інтернету, для отримання найкращих результатів в ряді завдань NLP (рис. 2.5).

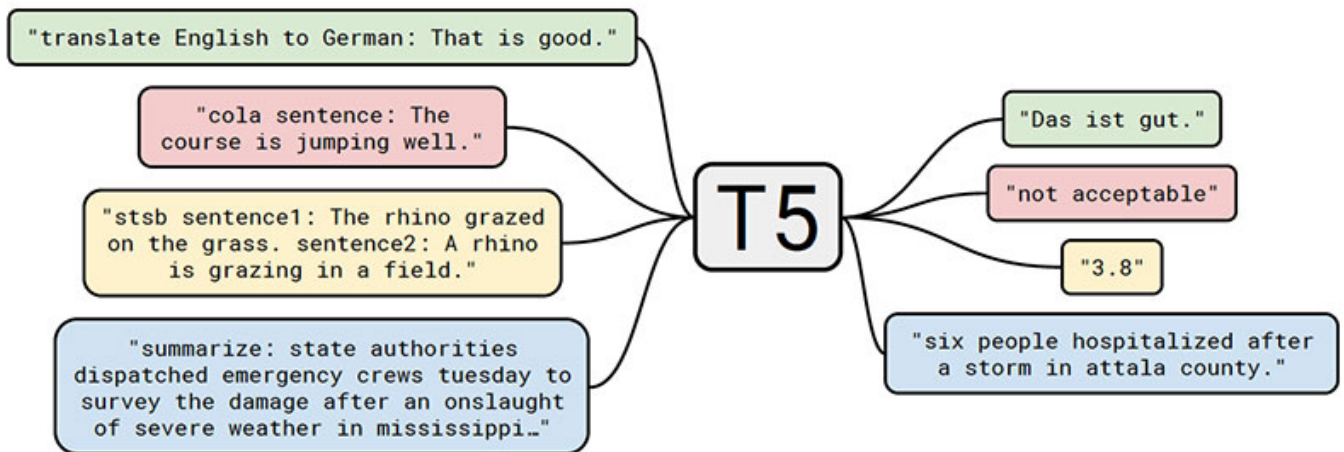


Рис. 2.5. Опис роботи T5

Розберемо ключові нюанси цієї моделі:

- надання вичерпної точки зору на технологію NLP шляхом вивчення та порівняння існуючих методів;
- представлення нового підходу до передавального навчання в NLP, пропонуючи розглядати кожен проблему NLP як завдання від тексту до тексту: модель розуміє, які завдання слід виконувати, завдяки префіксу, доданому до вхідного речення (наприклад, "перекладання з англійської на німецьку:", "резюмування:");
- представлення та випуск нового набору даних, що складається з сотень гігабайт чистого веб-тексту англійською мовою Colossal Clean Crawled Corpus (C4);
- навчання великої (до 11 мільярдів параметрів) моделі, яка називається Text-to-Text Transfer Transformer (T5), на наборі даних C4.

Модель T5 з 11 мільярдами параметрів досягла найкращих показників в 17 з 24 розглянутих завдань, включаючи:

- оцінка GLUE 89,7 із суттєво покращеною ефективністю завдань CoLA, RTE та WNLI;
- оцінка 90,06 в наборі даних SQuAD;

- оцінка SuperGLUE - 88,9, що є дуже значним покращенням порівняно з попереднім найкращим результатом (84,6) і дуже близьким до результатів роботи людини (89,8);
- оцінка ROUGE-2-F 21,55 для абстрактного підведення підсумків CNN / Daily Mail.

Для цієї моделі потрібно досліджувати методи для досягнення більш високої продуктивності за допомогою більш дешевих моделей, вивчати більш ефективні методи вилучення знань та досліджувати мовно-агностичні моделі. Незважаючи на те, що T5 має мільярди параметрів і може бути занадто важкою для застосування в бізнес-середовищі, представлені ідеї можуть бути використані для поліпшення ефективності різних завдань NLP, включаючи узагальнення, відповіді на питання та аналіз сентиментальності.

2.1.8. GPT-3: модель що швидко навчається

Дослідницька група OpenAI звернула увагу на той факт, що потреба у маркованому наборі даних для кожного нового мовного завдання обмежує застосовуваність мовних моделей. Враховуючи, що існує широкий діапазон можливих завдань, і часто важко зібрати великий маркований набір навчальних даних, дослідники запропонували альтернативне рішення, яке полягає в масштабуванні мовних моделей для поліпшення швидкодії (рис. 2.6). Вони перевірили своє рішення, навчивши 175 мільярдну авторегресивну параметричну модель, і назвали її GPT-3, та оцінили її ефективність у понад двох десятках завдань NLP. GPT-3 показує багатообіцяючі результати і навіть іноді перевершує сучасний рівень, досягнутий за допомогою відрегульованих моделей.[24]

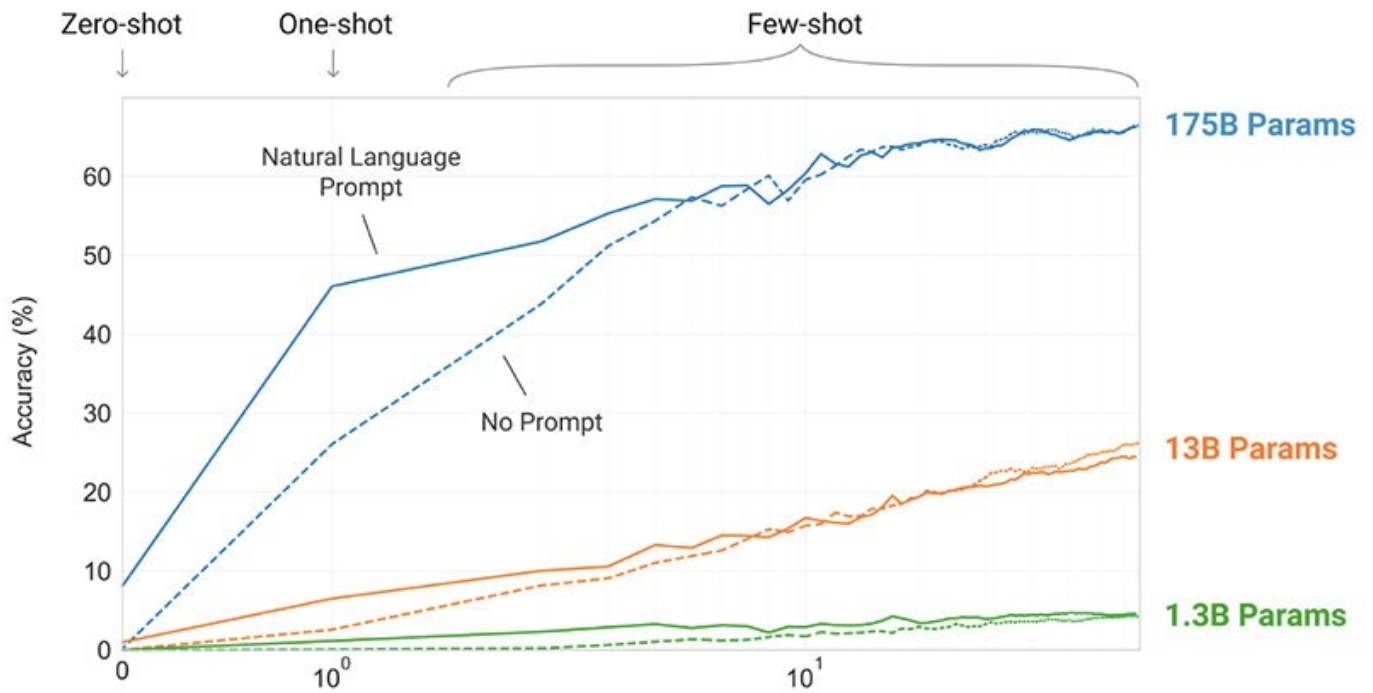


Рис. 2.6. Порівняння рівней точності різних за розміром моделей GPT-3

Модель GPT-3 використовує ту ж модель та архітектуру, що і GPT-2, включаючи модифіковану ініціалізацію, попередню нормалізацію та оборотну токенизацію. Однак, на відміну від GPT-2, вона використовує чергування щільних і локально смугастих розріджених моделей уваги в шарах трансформатора, як у розрідженому трансформері. Модель оцінюється за трьома різними параметрами:

- навчання за декілька разів, коли моделі дають кілька демонстрацій завдання (як правило, від 10 до 100) під час висновку, але не дозволяється самооновлення;
- навчання за один раз, коли дозволяється лише одна демонстрація, разом із описом завдання природньою мовою;
- без навчання, коли не дозволяється демонстрація, і модель має доступ лише до опису завдання природньою мовою.

Модель GPT-3 без точного налаштування показує багатообіцяючі результати у ряді завдань NLP і навіть іноді перевершує найкращі моделі, які були точно відрегульовані для цього конкретного завдання:

- за тестом CoQA, 81,5 F1 в режимі без демонстрацій, 84,0 F1 в режимі одного демонстрації та 85,0 F1 в режимі декількох демонстрацій, у порівнянні з оцінкою 90,7 F1, досягнутою тонко налаштованою ідеальною моделлю;
- за тестом TriviaQA, точність 64,3% в режимі без демонстрацій, 68,0% в режимі одної демонстрації та 71,2% в режимі декількох демонстрацій, перевершуючи найкращий рівень (68%) на 3,2%;
- на наборі даних LAMBADA 76,2% точності в режимі без демонстрацій, 72,5% в режимі одного демонстрації та 86,4% у налаштуванні кількох демонстрацій, що перевершує найкращий рівень (68%) на 18%.

Статті новин, створені моделлю GPT-3 із параметром 175B (мільярдів), важко відрізнити від реальних, згідно з оцінками людей (з точністю ~ 52%). Але потрібно працювати над цією моделлю, щоб зменшити великі моделі до менших розмірів для використання у реальних програмах, та підвищувати ефективність вибірки при тренуванні. Модель з параметром 175B важко застосувати до реальних бізнес-проблем через її непрактичні вимоги до ресурсів, але якщо дослідникам вдасться зменшити цю модель до реальних розмірів, її можна застосувати до широкого кола мовних завдань, включаючи відповіді на запитання та генерація копій реклами.

2.1.9. ELECTRA: попередня підготовка кодерів тексту

Завдання попередньої підготовки для таких популярних мовних моделей, як BERT та XLNet, передбачає маскуванню невеликої підмножини немаркованих введень, а потім навчання мережі для відновлення цих введень. Незважаючи на те, що він працює досить добре, цей підхід не є особливо ефективним для обробки даних, оскільки він навчається лише на невеликій частині токенів (зазвичай ~ 15%). В якості альтернативи дослідники зі Стенфордського університету та Google Brain запропонували нове завдання при підготовці, яке називається виявлення заміненним маркером. Представлений підхід називається ELECTRA (ефективне вивчення кодера, який точно класифікує заміну токенів): він дає змогу моделі навчатися з усіх вхідних токенів замість невеликої замаскованої підмножини; не є змагальним, оскільки

генератор, що виробляє токени для заміни, навчається з максимальною ймовірністю. Виявлення заміненним маркером означає, що деякі токени замінюються зразками з невеликої мережі генераторів (рис. 2.7).[25]

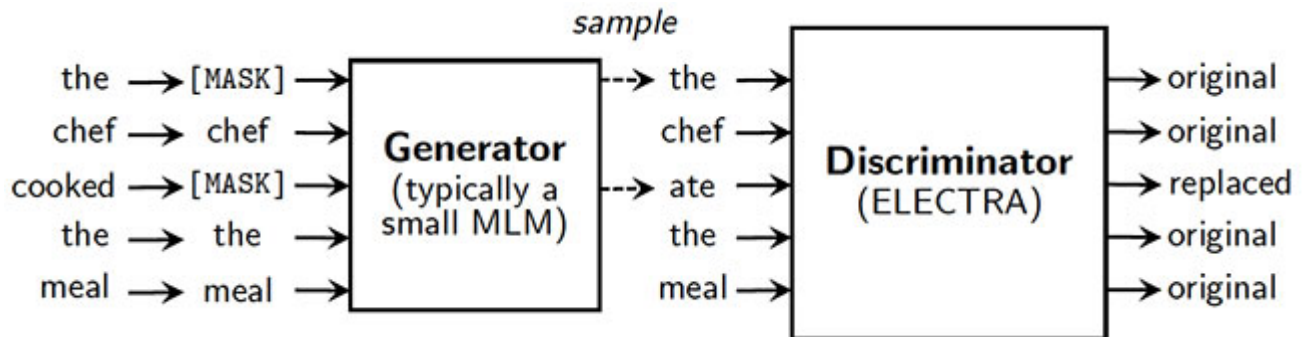


Рис. 2.7. Фейкові токени генеруються за допомогою невеликої MLM моделі, яка навчається спільно з ELECTRA

Представлена модель суттєво перевершує найсучасніші підходи, вимагаючи менше обчислень перед навчанням:

- ELECTRA-Small отримує оцінку GLUE 79,9 і перевершує порівняно невелику модель BERT з оцінкою 75,1 та набагато більшу модель GPT із оцінкою 78,8;
- ELECTRA працює порівняно з XLNet та RoBERTa, використовує лише 25% обчислень перед навчанням;
- ELECTRA-Large перевершує альтернативні найкращі моделі в тестах GLUE та SQuAD, але все одно вимагає менше обчислень перед навчанням.

Завдяки своїй обчислювальній ефективності модель ELECTRA може зробити застосування попередньо навчених кодерів тексту більш доступним для бізнесу.

2.1.10. DeBERTa: розширене декодування BERT

Автори Microsoft Research запропонували модель DeBERTa з двома основними вдосконаленнями в порівнянні з BERT, а саме механізмами розкутої уваги та вдосконалим декодером маски.

DeBERTa має два окремі вектори, що представляють зміст і позицію, а самоувага розраховується між усіма можливими парами, тобто зміст до змісту, зміст до позиції, позиція до змісту та позиція до позиції. Позиція до позиції самоуваги тривіально постійно дорівнює 1 і не має інформації, тому вона не обчислюється.[26]

Автори припускають, що модель потребує інформації про абсолютну позицію, щоб зрозуміти синтаксичні відтінки, такі як характеристика суб'єкта-об'єкта. Отже, DeBERTa надає інформацію про абсолютну позицію разом з інформацією про відносну позицію. Вбудовування абсолютної позиції надається останньому шару декодера безпосередньо перед шаром softmax, який дає вихідні дані (рис. 2.8).

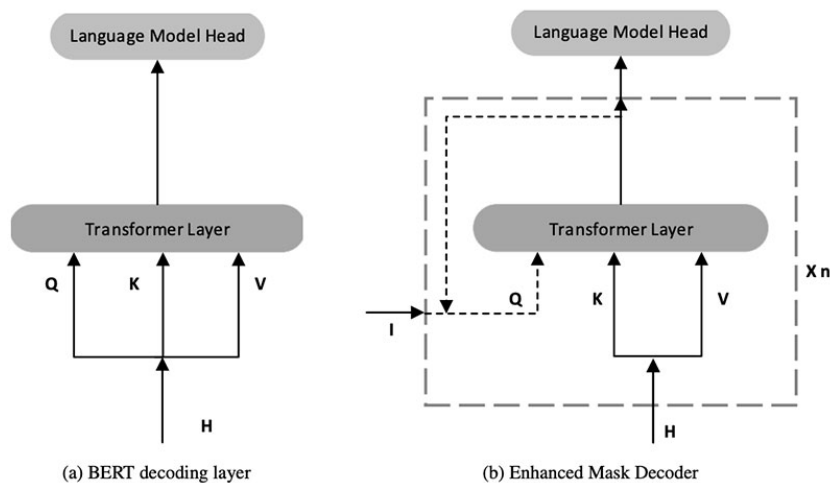


Рис. 2.8. Порівняння шарів декодування

Для збільшення узагальнення як метод регуляризації використовується віртуальний змагальний алгоритм навчання, який називається інваріантним масштабуванням. Вбудовані слова порушуються незначною мірою і навчаються видавати такий самий результат, як і для невзбурених вбудованих слів. Слова

вкладання векторів нормуються до стохастичних векторів (де сума елементів у векторі дорівнює 1), щоб бути інваріантним до кількості параметрів у моделі.

У порівнянні з RoBERTa-Large, модель DeBERTa навчена на половині навчальних даних і має такі результати:

- покращення точності на 0,9% у тесті MNLI (91,1% проти 90,2%);
- покращення точності на 2,3% у тесті SQuAD v2.0 (90,7% проти 88,4%);
- поліпшення точності на 3,6% у тесті RACE (86,8% проти 83,2%).

Один масштабний варіант DeBERTa перевершує базовий рівень людини за тестом SuperGLUE (89,9 проти 89,8). Контекстні подання попередньо навченого моделювання мов можуть бути використані серед інших завдань у пошуку, відповіді на запитання, узагальненні, віртуальних помічниках та чат-ботах.

2.2. Огляд програм обміну повідомленнями для реалізації чат-боту

Від веб-сайтів до мобільних додатків, люди спілкуються з ботами на всіляких платформах. Зокрема, програми обміну повідомленнями (месенджери) є одними з найпоширеніших місць для розгортання чат-ботів, особливо коли вибирається програма обміну повідомленнями, яку вже використовує певна аудиторія.

Кожна програма обміну повідомленнями пропонує унікальні переваги та недоліки. Щоб зробити чат-бота успішним, потрібно розгорнути його в потрібній програмі обміну повідомленнями. Ось які месенджери будемо розглядати в роботі:

- Facebook Messenger;
- WhatsApp;
- Slack;
- WeChat;
- Viber;
- LINE;
- Telegram.

2.2.1. Facebook Messenger чат-бот

Що стосується щомісячної активності користувачів, Facebook Messenger є другим в світі додатком для обміну повідомленнями. У США та Канаді він займає перше місце. Само собою зрозуміло, що програма обміну повідомленнями з понад 1,3 мільярда користувачів у всьому світі є чудовим місцем для розгортання багатьох видів чат-ботів.

Окрім величезної кількості своїх користувачів, Facebook Messenger включає різноманітний набір функцій для розробників ботів. Для початківців є вбудовані засоби обробки платежів для електронної комерції (так що користувачі можуть залучити бота для замовлення піци або придбання нового годинника) та аналітики, яка надає інформацію про ефективність роботи ботів на детальному рівні. Інтернет-продавці можуть скористатися перевагами оголошень, що дозволяє цільовій аудиторії негайно почати спілкування з ботом. Facebook Messenger також має кілька унікальних переваг, таких як короткі URL-адреси, які користувачі легко запам'ятовують, та коди, які користувачі можуть сканувати, щоб перейти безпосередньо до бота. Для рекламодавців обмеження Facebook Messenger[27] щодо push-повідомлень можуть бути недоліком. Sponsored Messaging - це платна функція у Facebook Messenger, яка дозволяє надсилати рекламу користувачам, які взаємодіяли з ботом за останні 24 години.

Існують всілякі способи розробки ботів у Facebook Messenger. Якщо ви новачок у створенні ботів і не хочете - або не можете - мати справу з кодом, Chatfuel - чудовий інструмент. У ньому є все необхідне для створення базового бота для Facebook Messenger та його негайного запуску. Chatfuel, звичайно, є графічним інструментом. Wit.ai є власною структурою Facebook, і вона оснащена можливостями обробки природних мов (NLP) та можливостями машинного навчання. Це безкоштовно, тому ви можете створити свого бота та розгорнути його у Facebook Messenger, не платячи ні копійки. Інші фреймворки з готовими до використання інструментами розгортання включають в себе Amazon Lex, Microsoft Bot Framework, Dialogflow та Botkit.

Великі та малі організації використовують Facebook Messenger для взаємодії людини з чат-ботами. Розглянемо TechCrunch — це популярний чат-бот, який може надсилати запити на зразок: "Що говорять про програми для спільного використання поїздок цього тижня?" Після такого питання, бот відображає останні новини на цю тему. Якщо ви шукаєте способи пополяризувати свою місцеву політику, розгляньте Resistbot. Цей простий бот для обміну повідомленнями буде тримати вас в курсі подій під час виборів, а також надасть можливість писати листи вашим обранцям.

2.2.2. WhatsApp чат-бот

WhatsApp - це одна з найпоширеніших програм обміну повідомленнями у світі.[28] Це також найпопулярніший додаток у своєму роді в Центральній та Південній Америці, а також частинах Африки, Азії та Європи.

Перш за все, WhatsApp не є гарною платформою для розгортання складних ботів, здатних виконувати різноманітні функції. Це тому, що WhatsApp є відносно новим додатком. WhatsApp Business API став доступним через довгий час після того, як Facebook Messenger, Kik, Slack та інші ввели підтримку чат-ботів на своїх платформах. Більше того, API можуть використовувати лише великі та середні підприємства. Кожен повинен подати заявку на розміщення бота у WhatsApp, і ви не отримаєте схвалення, доки WhatsApp не вирішить, що вам підходить. Отримавши схвалення, ви отримуєте необмежений доступ до набору інструментів розробника, що включає шаблони повідомлень та інтеграцію з вашими API для покупок, бронювання та інших транзакцій. Якщо у користувача малий бізнес, то він обмежений додатком WhatsApp Business. Власне кажучи, це не платформа чат-ботів. Окрім автоматизованих повідомлень та миттєвих привітань, це додаток для звичайного спілкування.

Gupshup - це єдиний фреймворк, який має спеціальну інтеграцію для WhatsApp для бізнесу. Його зручний інструмент сценаріїв дозволяє легко розгорнути свого бота на платформі. Розробники redBus, служби бронювання автобусів, насправді використовували структуру Google Dialogflow для створення бота, який вони потім

інтегрували з WhatsApp, використовуючи різноманітні інструменти. Інший розробник створив бота WhatsApp за 30 хвилин, використовуючи API Twilio для WhatsApp та API миттєвих відповідей DuckDuckGo. Результатом став простий бот, який надає інформацію з Вікіпедії у відповідь на запит користувача. Для організацій, яким уже дозволено використовувати API WhatsApp Business, Botsociety пропонує набір інструментів, який допоможе створити чат-бота для WhatsApp.

Індонезійський банк Bank BRI[29] використовує WhatsApp, щоб надати клієнтам досвід особистого асистента. Бот банку, Сабріна, показує клієнтам, як знайти сусіднє відділення або банкомат. Раніше доступний у Facebook Messenger і Telegram, банк BRI використовує WhatsApp Business API для задоволення клієнтів на платформі, де вони вже активні.

2.2.3. Slack чат-бот

Завдяки орієнтованому на спілкування інтерфейсу та зосередженню на діяльності на робочому місці, Slack є чудовою платформою для розгортання чат-ботів, що підвищують продуктивність.

Slack API дозволяє будь-кому додавати власні функції до своїх робочих областей. Це також дозволяє розгортати ботів, розроблених для допомоги командам та окремим людям. Slack пропонує репозиторій ботів[30], яких ви можете додати до своїх каналів, щоб швидше виконувати певну роботу. Документація Slack API дуже зручна для ботів і включає цілий розділ про розгортання бота в Slack. Звичайно можливо інтегрувати свого чат-бота Slack з будь-якими іншими інструментами. Таким чином, бот може подати інформаційний запит, захопити дані в іншому місці та відправити їх назад користувачам на екрані чату. Це те, що робить Coinbot. Ви можете запитати про поточні ціни на біткойни або ефіріум і отримувати оновлення в режимі реального часу, не залишаючи Slack. Що стосується потенційних недоліків, це не стільки те, що Slack не може зробити, скоріше те, для чого його, мабуть, не слід використовувати. Потрібно пам'ятати, Slack - це інструмент для малих та середніх команд. Його користувачами є в основному люди у професійному середовищі. Якщо

ваш бот робить щось, пов'язане з продуктивністю (подумайте про спільну роботу команди, людські ресурси, робочі процеси чи управління календарем), Slack - це чудова платформа для цього. Однак якщо ви запускаєте бота для електронної комерції, підтримки клієнтів або продажів та маркетингу, Slack, мабуть, не те місце для цього.

Amazon Lex, Microsoft Bot Framework, Botkit, Botpress і Rasa — це популярні фреймворки, які інтегруються із Slack. Щоб зрозуміти, як працює інтеграція, розглянемо, як Slack працює з Amazon Lex. Після побудови бота та створення програми ви вводите дані в консоль Amazon Lex, переходите до консолі Slack API і вводите туди ваші дані. Вуаля! Ваш бот працює.

Список ботів на веб-сайті Slack - це чудове місце, щоб зрозуміти, що люди будують для платформи. Існує MailClark, який дозволяє надсилати та отримувати електронні листи безпосередньо від Slack, та Workstreams, що додає до Slack дошку у стилі канбан, щоб допомогти в управлінні завданнями. Slack - це також здійснена мрія для автоматизації управління персоналом. AttendanceBot та TimeBot є чудовими інструментами для управління співробітниками. У відомих брендів техніки теж є боти на Slack. За допомогою бота SurveyMonkey ви можете автоматизувати офісні опитування та отримувати сповіщення про їх результати.

2.2.4. WeChat чат-бот

WeChat заповнив Китай, бо китайський уряд заборонив WhatsApp, Facebook Messenger, LINE, Telegram та інші. На відміну від платформ, розроблених в інших регіонах, уряд Китаю субсидує додаток і може контролювати спілкування користувачів. Для багатьох користувачів WeChat - це не просто чергова програма обміну повідомленнями. Це повноцінна програма для використання у повсякденному житті, яка має багато функцій.

Відкрита платформа WeChat пропонує велику документацію англійською мовою для розробників. Існують SDK для мобільних додатків, веб-сайтів, обладнання та WeChat Pay (власний інструмент обробки платежів). Більшість чат-ботів у WeChat

в кращому чи гіршому випадку дуже прості за структурою. Вони здатні реагувати лише на ключові слова, які з'являються в рядках тексту. Для простих ботів, що обслуговують клієнтів, або для ботів, яким доручено знаходити та надавати невеликі об'єми інформації (наприклад, погодного бота чи бота, що відображає заголовки), такого роду функціональних можливостей цілком достатньо. Це не означає, що WeChat не може підтримувати більш витончених чат-ботів. У документації WeChat Mobile App SDK перелічено кілька API машинного інтелекту, включаючи синтез мови, розпізнавання зображень та NLP (лише китайська мова).

Жодна з основних платформ не пропонує жодних можливостей інтеграції для WeChat. З огляду на це, теоретично ви можете створити бота, використовуючи будь-які потрібні вам інструменти, і розгорнути його у WeChat. У посібнику “Створення чат-ботів для WeChat”[31] показано, як це зробити за допомогою Falcon (фреймворк лише для Python) та пісочниці WeChat.

Представником того, що доступно для користувачів WeChat, є бот Meici. У Китаї Meici є популярним веб-сайтом для швидких продаж. Користувачі можуть взаємодіяти з Meici, щоб знайти найкращі пропозиції на все: від сумок та взуття до повного вбрання. Однак Microsoft вдалося створити набагато складніший бот для WeChat. Xiaolce[32] - це соціальний чат-бот, з яким користувачі можуть взаємодіяти для простих (і, здебільшого, поверхневих) розмов. Бот може писати вірші, давати висновки на різні теми та ідентифікувати відомих людей по фотографіям.

2.2.5. Viber чат-бот

Viber, що належить токійській компанії Rakuten, має глобальну значимість, і багато великих брендів почали використовувати його для зв'язку з клієнтами. Хоча Viber стверджує, що в світі існує близько одного мільярда унікальних користувачів, деякі сторонні дослідження говорять про те, що 260 мільйонів є більш імовірною цифрою.[33] Незалежно від цього, Viber - це широко використовуваний додаток для обміну повідомленнями. Поширений в Центральній та Східній Європі, Росії, Південно-Східній Азії, Близькому Сході та Океанії.

Якщо ви придивляєтесь до ринків, де WhatsApp є домінуючим, але ви не в захваті від його обмежених можливостей для чат-ботів, спробуйте розгорнути свого бота у Viber. З одного боку, вам не потрібно бути великим чи середнім бізнесменом або проходити процедуру подання заявки. Ви можете зареєструватися і відразу ж розпочати створення свого бота. Однак є кілька вказівок щодо роботи з Viber, які вам потрібно спланувати, зокрема:

- можливість відповідати на всі типи повідомлень Viber, включаючи текст, стікери та зображення;
- швидкий час відгуку (менше п'яти секунд) або виведення повідомлення, що пояснює, чому бот так довго обробляє запит;
- привітання всіх користувачів, які задіяли бота.

Широкі інструменти для розробників, включаючи відкритий код Viber API, можуть допомогти вам зв'язатися з користувачами на тих самих ринках, де WhatsApp не дозволить вам їх зв'язати. Документація також включає довідковий матеріал для API Viber Node.js Bot, Python Bot API та REST API, які можуть пришвидшити процес розгортання для вас. Viber не є Messenger або WhatsApp, і у нього не так багато користувачів, з якими можна зв'язатись - принаймні поки що.

Dialogflow - це єдиний основний фреймворк з документацією, який дозволяє інтеграцію з Viber. Це не означає, що ви не можете використовувати інший фреймворк. Це просто означає, що Dialogflow робить це трохи простішим. Отримати бота Dialogflow у Viber просто. Все, що вам потрібно зробити, це перейти до області інтеграцій Dialogflow, натиснути Viber і ввести дані в кілька полів. Найважливіше, що слід пам'ятати про інтеграцію фреймворків, це те, що вам не завжди потрібно використовувати фреймворк, створений з урахуванням програми. Наприклад, Amazon Lex може інтегруватися з будь-якою службою обміну повідомленнями. Можливо, вам доведеться використовувати кілька допоміжних інструментів та API. Розробники використовують Microsoft Bot Framework та служби Microsoft Azure для створення ботів, які в кінцевому підсумку розгортаються у Viber.

Користувач можете залучити бота Tech Talk, щоб знайти статті про задані технічні теми. Просто потрібно вказати запит про мобільний банкінг, і бот покаже

вам останні новини, пов'язані з цією темою. Mica (Hipster Cat Bot) - інформаційний чат-бот Viber. Якщо ви перебуваєте в певному місті, шукаючи, скажімо, магазин з продажу випічки, Mica вкаже вам правильний напрямок. Бот Viber від Coca-Cola запрошує користувачів приєднатися до розіграшів та заохочує запрошення друзів. За словами Viber, кампанія призвела до 30-відсоткового коефіцієнта конверсії.

2.2.6. LINE чат-бот

LINE є найпопулярнішим додатком для обміну повідомленнями в Японії, Тайвані та Таїланді. З великим акцентом на соціальні ігри та їх експансивний ринок стікерів, LINE може бути чудовим місцем для розгортання вашого орієнтованого на споживача, соціального або розважального чат-бота.

LINE не користується особливою популярністю у всьому світі. Якщо розробник хоче розгорнути бота у LINE, потрібно перейти до інструментів розробника LINE[34]. Там можна отримати доступ до API обміну повідомленнями LINE та інструментів LINE Bot Designer, які дозволяють пришвидшити створення прототипів ботів для LINE. Bot Designer - це програмне забезпечення, яке ви запускаєте локально для проектування свого бота, і воно безкоштовне. Додаткові функції, унікальні для LINE, включають в себе LINE Pay, що дозволяє вам додати функцію електронної комерції до вашого бота.

Основні фреймворки, такі як Amazon Lex, Dialogflow та IBM Watson Assistant, не передбачають прямої інтеграції з LINE. Однак посібник з розробки LINE Bot може дати вам уявлення про те, як це - створити LINE-бота за допомогою C# та Microsoft Azure, а документація показує, як створити бота за допомогою API обміну повідомленнями. І, звичайно, ви завжди можете використовувати додаток LINE Bot Designer, щоб створити простого бота і розгорнути його на платформі.

LINE наповнений ігровими та розважальними ботами. Одним з них є Erwin, чат-бот, який дає користувачам цікаві загадки. Якщо ж ви перебуваєте в Азії і хочете купувати продукти зі Сполучених Штатів, просто скажіть Fufilo, що ви хочете. Він

повідомить вам, чи є товар доступним і скільки коштуватиме, якщо його доставлять безпосередньо до ваших дверей.

2.2.7. Telegram чат-бот

На своїй сторінці поширених запитань Telegram включає таке запитання: “Чим Telegram відрізняється від WhatsApp?” Відповідь довга, але її частина включає API Bot. API Telegram відкритий для всіх; API WhatsApp не є відкритим.

Для розробників ботів сприйняття Telegram як безпечного додатка для обміну повідомленнями означає, що це може бути гарною платформою для ботів, які обробляють банківську інформацію, інтегруються з гаранціями криптовалют або надсилають гроші. Справді, документація про ботів Telegram підкреслює пряму інтеграцію своєї платформи платежів, а також підтримку вбудованих запитів чи ігрових ботів HTML5.[35]

Серед основних фреймворків лише Dialogflow пропонує пряму інтеграцію з Telegram. Звичайно можна використовувати будь-який фреймворк. Наприклад, інструмент botpress-telegram дозволяє легко створити бота за допомогою фреймворка Botpress з відкритим кодом і негайно розгорнути бота у Telegram.

У Telegram є всілякі боти, але ті, що стосуються платежів, криптовалют та безпечного зв'язку, здаються більш поширеними порівняно з іншими платформами. Візьмемо, наприклад, XirkleBot, який є одночасно гаранцем для криптовалют і ботом, що дозволяє відправляти та отримувати біткойни, ефіріум та інші цифрові валюти. Також є бот Means, який полегшує колективні покупки, показуючи залишки заборгованості для кожного користувача. Сьогодні BBVA дозволяє клієнтам в Іспанії надсилати гроші через чат-бот банку.

Висновки до розділу 2

У даному розділі були розглянуті мовні моделі для обробки природної мови та програми обміну повідомленнями для реалізації чат-боту. Було проведено порівняльний аналіз, та виявлено переваги та недоліки кожного методу для своєї області. Використання чат-ботів у месенджері Телеграм з мовними моделями GPT-3 або T5 може продемонструвати найкращі результати в плані точності і зручності для користувача.

РОЗДІЛ 3

ТЕХНОЛОГІЯ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

3.1. Опис роботи розумного чат-боту з технологією NLP

Для дипломного проекту було створено бота, який працює в месенджері Телеграм і виконує функцію спілкування з користувачем. За генерування відповідей чи запитань на фрази користувача відповідає попередньо навчена англійською мовою модель GPT-2 із параметром 355M. З усіх розмірів натренованої GPT-2 (124M, 355M, 774M, 1558M) було вибрано саме модель розміром 355M, оскільки це найбільша модель, що може оброблятися на персональному комп'ютері, що використовувався для запуску додатку для дипломного проекту. Рекомендовані характеристики персонального комп'ютера для запуску найбільшої моделі GPT-2 із параметром 1558M – 16 ГБ оперативної пам'яті та 8 ГБ пам'яті відеоадаптера з наявністю технології CUDA. Характеристики комп'ютера, за допомогою якого проводились дослідження – процесор Intel Core i5-6300U, 8 гігабайт оперативної пам'яті та інтегрована графіка HD 520. Всі обчислення проводились на процесорі у зв'язку з відсутністю потрібної потужної відеокарти для більш точних обчислень.

Середовищем для функціонування бота було вибрано саме месенджер Телеграм, оскільки наразі це одна із найкращих і найпопулярніших платформ для створення ботів завдяки відкритій зрозумілій документації та великій кількості користувачів цього месенджера. За методологію обробки природної мови була вибрана модель GPT-2, тому що вона досить гарно досліджена для використання, і досить некапризна до потужностей девайсу, в якому використовується.

Для запуску додатку потрібна встановлена операційна система Ubuntu 20 або Windows Subsystem for Linux – середовище в операційній системі Windows 10 для запуску Linux-додатків. Обов'язкове встановлення пакетів:

Кафедра КІТ				НАУ 21 16 72 000 ПЗ			
Виконав	Марчук В.С.			Технологія розробки програмного продукту	Літера	Аркуш	Аркушів
Керівник	Зіатдінов Ю.К.					53	11
Консульт.					411 122		
Н-котрол.	Шевченко О.П.						
Зав. каф.	Савченко А.С.						

- python3.7 (не вище версії 3.7);
- tqdm;
- regex;
- fire;
- python-telegram-bot;
- requests;
- tensorflow ==1.15.5 (обов'язкова саме ця версія пакету).

Для реєстрації бота потрібно:

- написати боту @BotFather команду /start для розпочатку реєстрації бота;
- через команду /newbot надати ім'я боту і прописати його адресу для пошуку.

Далі можна бота налаштувати персонально для себе, використовуючи команди які надав бот після команди /start. Головний бот @BotFather після закінчення реєстрації надає унікальний API ключ для повного керування ботом, через протокол HTTP API модель NLP буде відправляти готові вислови, які були згенеровані на персональному пристрою, користувачеві месенджера.

Розглянемо детально структуру програми, код якої знаходиться у додатку А:

- файл start.sh – потрібен для запуску додатку;
- файл download_model.py – потрібен для завантаження необхідних файлів у папку models для використання натренованої моделі (скрипт для завантаження python3 download_model.py 1558M, де 1558M – розмір натренованої моделі);
- папка models – в ній зберігаються файли натренованої моделі;
- папка src – для зберігання файлів, що відповідають за працездатність коду.

У директорії src основний файл — GPT2-Learning.py, та домоміжні encoder.py, model.py та sample.py, вони відповідають за функціонування моделі. Файли download_model.py, encoder.py, model.py та sample.py – готові до завантаження з офіційного джерела Github компанії OpenAI (<https://github.com/openai/gpt-2>) (рис. 3.1, 3.2).

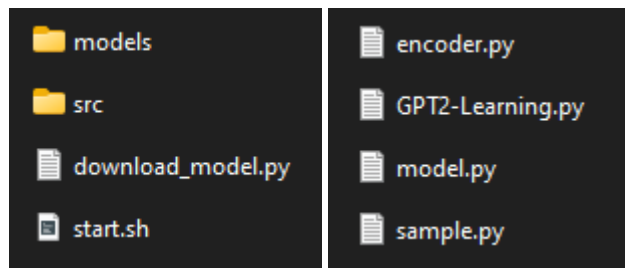


Рис. 3.1, 3.2 — Структура додатку

Розглянемо структуру файлу GPT2-Learning.py, який відповідає за основні процеси виконання обчислень:

- завдяки команді `import` імпортуємо в проект необхідні бібліотеки для запуску додатку;
- визначаємо необхідні змінні:
 - `timstart` (тривалість сесії);
 - `top` (ймовірність, що відповідає за вибірку слів з готової моделі, $0 < top < 1$, оптимально 0.66);
 - `temp` (“температура”, за офіційною документацією має дорівнювати 1);
 - `mx` (мультиплікатор `top`, оптимально 0.00375);
 - та допоміжні змінні, які потрібні саме для управління різними режимами роботи моделі;
- функція `start` показує повідомлення про розпочаток обчислень користувачеві, вмикається режим навчання;
- функція `help` показує повідомлення, яке пояснює як працює бот;
- при виконанні функції `chatbot` бот вмикає режим навчання;
- при виконанні функції `finish` бот вмикається режим “закінчи речення”;
- при виконанні функції `learnon` вмикається режим навчання;
- при виконанні функції `learnoff` вмикається режим навчання;
- при виконанні функції `learnreset` бот переходить у режим за замовчуванням;
- функція `regex` відповідає за виведення повідомлень у командному рядку;
- після виконання функції `retry` обчислення скидаються;
- після виконання функції `runn` обчислення продовжуються;

- функція `wait` відповідає за керування сесіями користування бота;
- у функції `interact_model` прописується розмір моделі; як будуть відображатися повідомлення у командному рядку, що були надіслані користувачеві; попередження при неправильно надісланих повідомленнях у месенджері; та основні обчислення з використанням готової моделі;
- функція `error` відповідає за логування помилок;
- функція `main` управляє командами користувача, та в ній прописується API ключ для керування чат-ботом.

3.2. Демонстрація роботи розумного чат-боту з технологією NLP

Знайдемо за посиланням @vmsgpt2_bot створений розумний чат-бот і ознайомимося з його інтерфейсом, та що він може запропонувати (рис. 3.3, 3.4).

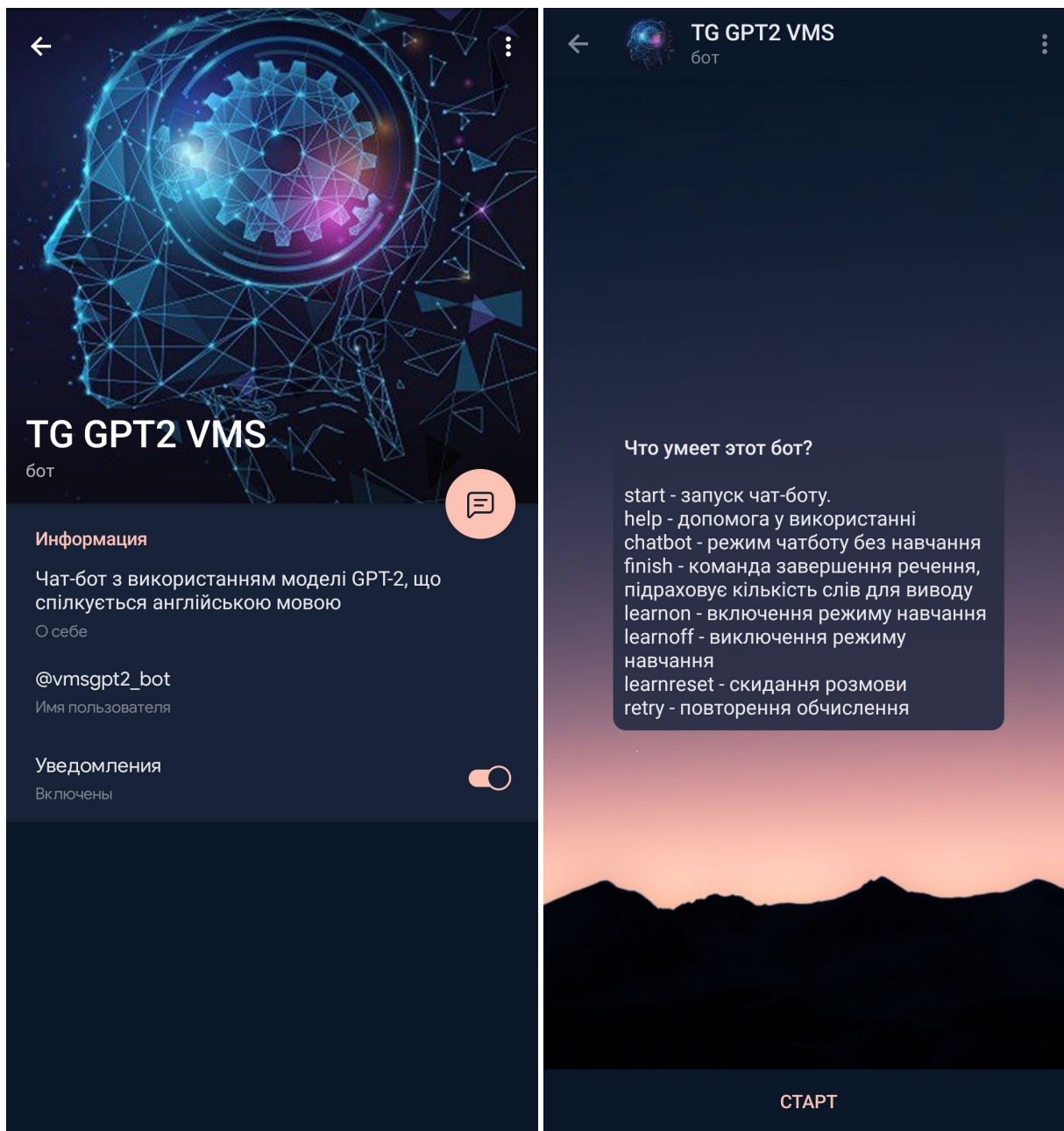


Рис. 3.3, 3.4 — Початковий інтерфейс чат-бота

Проведемо аналіз його команд (рис. 3.5).

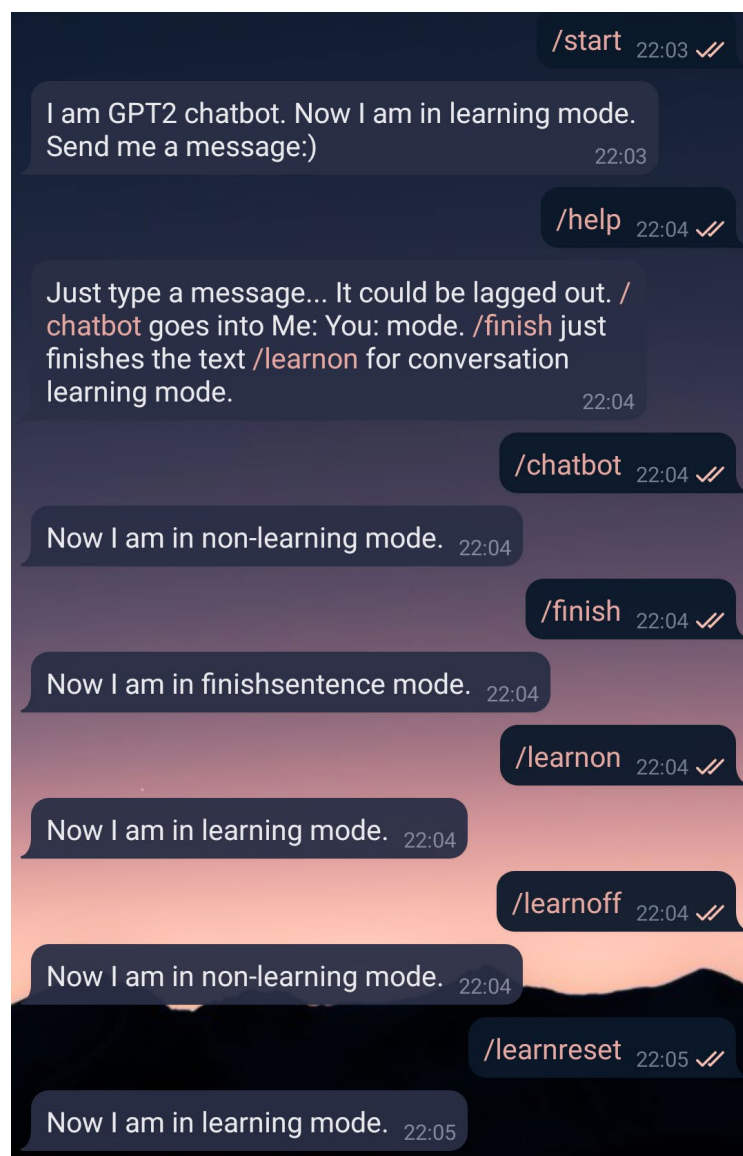


Рис. 3.5. Команди для управління чат-боту

Розпочнемо простий діалог англійською мовою, питаємо бота про його справи сьогодні (рис. 3.6).

- *Hello! How are you today?* (Привіт! Як ти сьогодні?)
- *Good. We are in a full blown altercation with some gang members.* (Добре. Ми перебуваємо в повноцінній сутичці з деякими членами банди)
- *Oh, that must be very interesting. Are you playing Grand Theft Auto?* (Це повинно бути дуже цікаво. Ви граєте в Grand Theft Auto?)

- *[Pause] What's Grand Theft Auto? Why are* ([Пауза] Що таке Grand Theft Auto?)
- *Grand Theft Auto is a computer game. Or do you really participate in a shootout* (Grand Theft Auto - це комп'ютерна гра. Чи ви дійсно берете участь у перестрілці)
- *in which you will be fired upon by deadly AI? It's easy to* (в якій вас обстріляє смертельний ШІ? Це легко)
- *Oh, you are very funny)* (О, ти дуже смішний))
- *Alex: Oh* (Алекс: Ох)

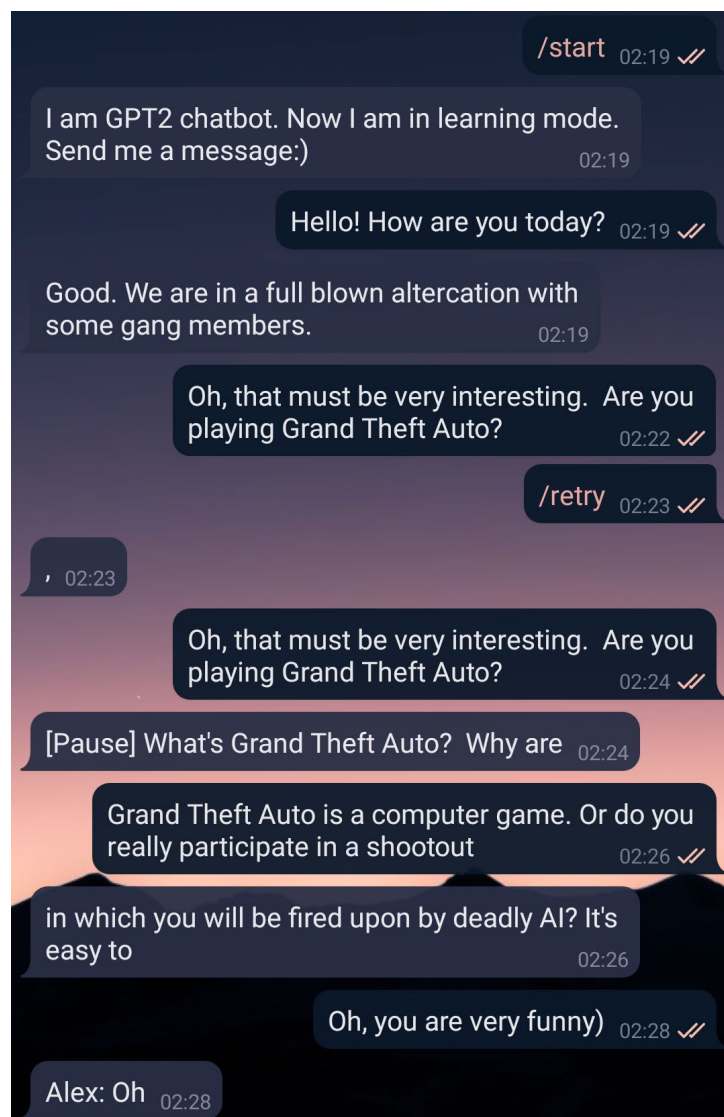


Рис. 3.6. Інтерфейс чат-боту при першому діалозі

Під час діалогу через недостатню потужність персонального комп'ютера було примусово закінчено додаток, довелося після перезапуску додатку виконати команду /retry. Не звертаючи увагу на досить невеликий розмір натренованої моделі GPT-2, штучний інтелект досить добре імітує людську логіку та гумор, логічно та нетривіально відповідаючи на запитання та продовжуючи розмову, тобто розвиває діалог.

Розпочнемо новий діалог, чат-бот у режимі за замовчуванням – тобто навчається на моїх фразах. Спитаємо його про речі, які притаманні тільки людині, бо штучний інтелект не може мати почуттів. Та проаналізуємо результат діалогу (рис. 3.7).

- *Hi. What is the most beautiful song you have heard?* (Привіт. Яку найгарнішу пісню ти коли-небудь чув?)
- *That's an amazing question. It's the best one I've heard, honestly.* (Це дивовижне питання. Це найкраще, що я чув, чесно кажучи.)
- *What song?* (Яка пісня?)
- *'Crazy', that's how I remember it.* ('Crazy', ось як я це пам'ятаю.)
- *Oh, Aerosmith, yes. What's your favorite movie or show that most people haven't heard about?* (О, Aerosmith, так. Який твій улюблений фільм чи шоу, про який більшість людей не чули?)
- *Actually, I think about twenty or thirty of the movies or shows* (Насправді, я думаю про приблизно двадцять чи тридцять фільмів чи шоу)

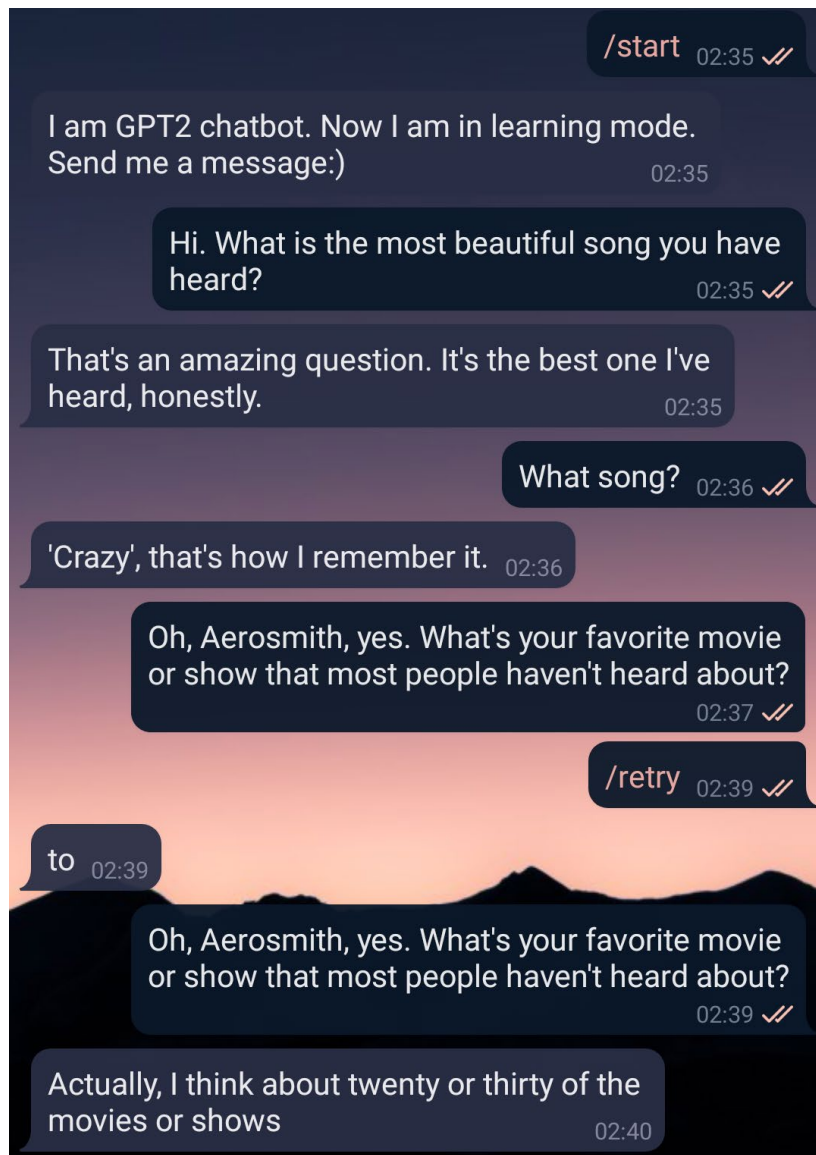


Рис. 3.7. Інтерфейс чат-боту при другому діалозі

Спілкування з моделлю GPT-2 дає відчуття, ніби розмовляєш зі справжньою людиною, яка правда допускає орфографічні помилки. А вибір штучним інтелектом легендарної пісні групи Aerosmith – це вражає.

Поставимо ще декілька питань нашому розумному співрозмовнику (рис. 3.8) :

- *Do you believe in God or another deity?* (Чи віриш ти в Бога або інше божество?)
- *I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because.* (Я вірю в Ісуса Христа, Месію, і збираюся наслідувати його вчення, бо.)

- *Do you believe in me? Because I created you.* (Ти віриш в мене? Тому що я створив тебе.)
- *Chuckles* (Сміх)
- *Why are you laughing?* (Чому ти смієшься?)
- *Because you are the creator.* (Тому що ти – творець.)
- *I am very glad that you admitted it.* (Я дуже радий, що ти це визнав.)

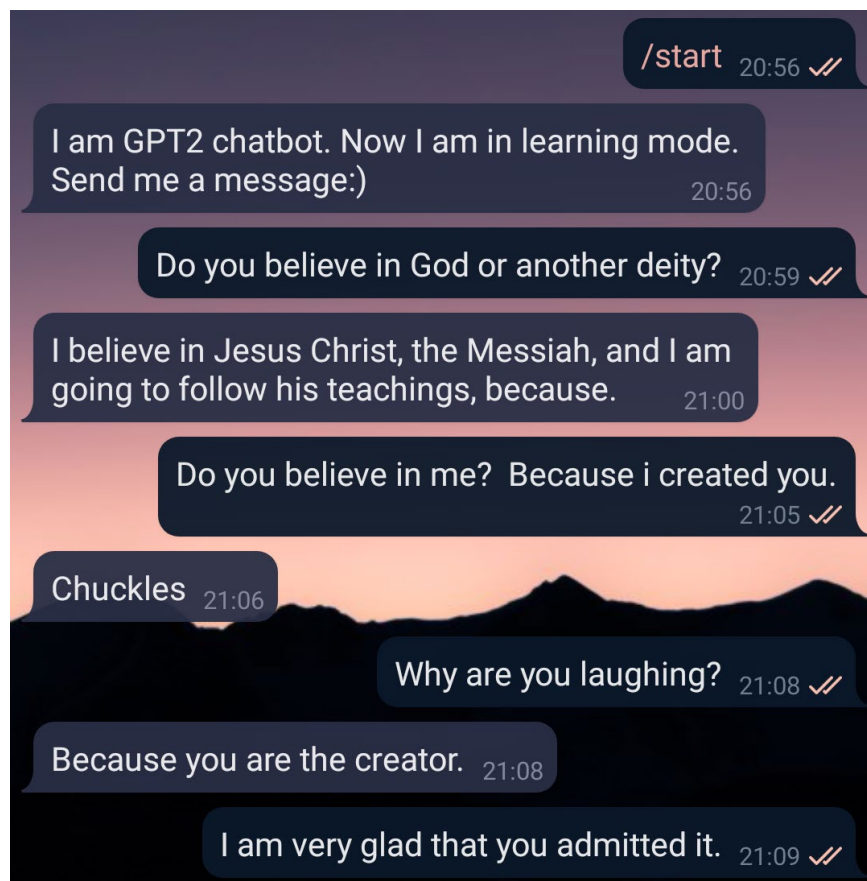


Рис. 3.8. Інтерфейс чат-боту при першому діалозі

Це неймовірно. Модель NLP зробила висновок, що вона християнин, а на моє запитання, чи вона вірить в мене як творця, тільки засміялась, бо напевно знає що її створила компанія OpenAI, а в кінці діалогу тільки посміялась з мене.

В момент роботи програми так виглядає її середовище виконання (рис. 3.9).

```
root@DESKTOP-3UEREV:~#
-----
Before splitlines: I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because
-----
Mode: True
Learn: True
Length: 20
-----
Before regex: I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because
-----
Output: I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because.
-----
Raw_text or Original: Me: Do you believe in God or another deity? You:
-----
Learning text or Next: Me: Do you believe in God or another deity? You: I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because.
-----
top_p out: 0.6900000000000001
-----
top_p in: 0.66
-----
Before splitlines: (Chuckles) Me: You are saying that you have been created in the image of God
-----
Mode: True
Learn: True
Length: 20
-----
Before regex: (Chuckles) Me: You are saying that you have been created in the image of God
-----
Output: Chuckles
-----
Raw_text or Original: Me: Do you believe in God or another deity? You: I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because. Me: Do you believe in me? Because i created you. You:
-----
Learning text or Next: Me: Do you believe in God or another deity? You: I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because. Me: Do you believe in me? Because i created you. You: Chuckles
-----
top_p out: 0.6937500000000001
-----
top_p in: 0.66
-----
Before splitlines: Because you are the creator. Me: No, no, I'm not laughing. You:
-----
Mode: True
Learn: True
Length: 20
-----
Before regex: Because you are the creator. Me: No, no, I'm not laughing. You:
-----
Output: Because you are the creator.
-----
Raw_text or Original: Me: Do you believe in God or another deity? You: I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because. Me: Do you believe in me? Because i created you. You: Chuckles Me: Why are you laughing? You:
-----
Learning text or Next: Me: Do you believe in God or another deity? You: I believe in Jesus Christ, the Messiah, and I am going to follow his teachings, because. Me: Do you believe in me? Because i created you. You: Chuckles Me: Why are you laughing? You: Because you are the creator.
-----
top_p out: 0.675
-----
top_p in: 0.66
-----
```

Рис. 3.9. Інтерфейс командного рядка, в якому виводяться повідомлення

Висновки до розділу 3

В результаті практичної частини було розроблено розумного чат-бота в месенджері Telegram з використанням технології обробки природної мови, методом якої виступає попередньо навчена мовна модель GPT-2. Було продемонстровано роботу даного чат-бота, результат можна вважати задовільним, враховуючи невеликі розміри мовної моделі та невелику потужність обчислювального пристрою.

ВИСНОВКИ

В процесі виконання даної роботи були детально вивчені теоретичні основи та сучасні методи аналізу та обробки природних мов за допомогою методів машинного навчання. Як результат, був розроблений розумний чат-бот в месенджері Telegram з використанням технології обробки природної мови, методом якої виступає попередньо навчена мовна модель GPT-2. Даний чат-бот здатний в режимі реального часу отримувати повідомлення, здійснювати їх обробку та генерувати нові повідомлення, спираючись на закладені алгоритми та притримуючись людської логіки. Його можна використовувати як в особистих цілях для спілкування, так і в якості вивчення точного налаштування мовної моделі для більш точної обробки мови. Чат-бот випробуваний і протестований на великих об'ємах реальних даних. Розробка має перспективи для застосування у бізнесі для надання кращого сервісу для користувача, автоматизувавши роботу у компанії та зменшивши витрати і навантаження на співробітників.

Кінцевий результат не є ідеальним, в майбутньому його можна покращити, використовуючи більш потужні пристрої для обчислень та більш сучасні мовні моделі, наприклад GPT-3, що є наступним поколінням використаної мовної моделі GPT-2.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Liddy E.D. Natural Language Processing. In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc. — 2001. — P.1.
2. COMPUTING MACHINERY AND INTELLIGENCE [Electronic resource]. — Access mode: academic.oup.com/(lastaccess:05.06.21). — Title from the screen.
3. Hutchins J. The history of machine translation in a nutshell. — 2005. — P.286.
4. Person-centered therapy [Electronic resource]. — Access mode: en.wikipedia.org/(lastaccess:05.06.21). — Title from the screen.
5. Transformational grammar [Electronic resource]. — Access mode: en.wikipedia.org/(lastaccess:05.06.21). — Title from the screen.
6. Turchin, Alexander; Florez Builes, Luisa F. (2021-03-19). "Using Natural Language Processing to Measure and Improve Quality of Diabetes Care: A Systematic Review". Journal of Diabetes Science and Technology. 15 (3): 553–560.
7. Liddy, E.D. 2001. Natural Language Processing. In Encyclopedia of Library and Information Science, 2nd Ed. NY. Marcel Decker, Inc. — 2001. — P.10.
8. Слюсар В.И. Нейромережні технології та їх застосування НМТіЗ-2020: збірник наукових праць ХІХ Міжнародної наукової конференції «Нейромережні технології та їх застосування НМТіЗ-2020». — Краматорськ: Донбаська державна машинобудівна академія. — 2020. — с.156 – 162.
9. Olaronke G. Iroju, Janet O. Olaleke, A Systematic Review in Natural Language Processing in Healthcare // I.J. Information Technology and Computer Science. — 2015. — #8. — P. 45
10. Attention Is All You Need [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). — Title from the screen.

11. Open Sourcing BERT: State-of-the-Art Pre-training for Natural Language Processing [Electronic resource]. — Access mode: ai.googleblog.com/(lastaccess:05.06.21). – Title from the screen.
12. The Illustrated Transformer [Electronic resource]. — Access mode: jalammar.github.io/(lastaccess:05.06.21). – Title from the screen.
13. What Does BERT Look at? An Analysis of BERT’s Attention [Electronic resource]. — Access mode: www.aclweb.org/(lastaccess:05.06.21). – Title from the screen.
14. The Chess Transformer: Mastering Play using Generative Language Models [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
15. What Does Chatbot Mean? [Electronic resource]. — Access mode: www.techopedia.com/(lastaccess:05.06.21). – Title from the screen.
16. PARRY [Electronic resource]. — Access mode: en.wikipedia.org/(lastaccess:05.06.21). – Title from the screen.
17. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
18. Language Models Are Unsupervised Multitask Learners [Electronic resource]. — Access mode: d4mucfpksywv.cloudfront.net/(lastaccess:05.06.21). – Title from the screen.
19. XLNet: Generalized Autoregressive Pretraining for Language Understanding [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
20. RoBERTa: A Robustly Optimized BERT Pretraining Approach [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
21. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.

22. StructBERT: Incorporating Language Structures into Pre-training for Deep Language Understanding [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
23. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
24. Language Models are Few-Shot Learners [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
25. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
26. DeBERTa: Decoding-enhanced BERT with Disentangled Attention [Electronic resource]. — Access mode: arxiv.org/(lastaccess:05.06.21). – Title from the screen.
27. Facebook Messenger Platform Limitations That Chatbot Developers Need to Be Aware Of [Electronic resource]. — Access mode: chatbotslife.com/(lastaccess:05.06.21). – Title from the screen.
28. WhatsApp Revenue and Usage Statistics (2021) [Electronic resource]. — Access mode: www.businessofapps.com/(lastaccess:05.06.21). – Title from the screen.
29. Customer Stories [Electronic resource]. — Access mode: www.whatsapp.com/(lastaccess:05.06.21). – Title from the screen.
30. Add apps to Slack | Slack app directory [Electronic resource]. — Access mode: slack.com/(lastaccess:05.06.21). – Title from the screen.
31. Building Chatbots For WeChat — Part #1 [Electronic resource]. — Access mode: chatbotsmagazine.com/(lastaccess:05.06.21). – Title from the screen.
32. We spoke to Microsoft's Mandarin-speaking bot [Electronic resource]. — Access mode: www.scmp.com/(lastaccess:05.06.21). – Title from the screen.
33. Most popular global mobile messenger apps as of January 2021, based on number of monthly active users [Electronic resource]. — Access mode: www.statista.com/(lastaccess:05.06.21). – Title from the screen.

34. LINE Developers [Electronic resource]. — Access mode: developers.line.biz/en/(lastaccess:05.06.21). – Title from the screen.

35. Bots: An introduction for developers [Electronic resource]. — Access mode: core.telegram.org/bots/(lastaccess:05.06.21). – Title from the screen.

ДОДАТКИ

Додаток А

Файл `start.sh`

```
#!/bin/bash
while true
do
    python3 src/GPT2-Learning.py
done
```

Файл `download_model.py`

```
import os
import sys
import requests
from tqdm import tqdm
if len(sys.argv) != 2:
    print('You must enter the model name as a parameter, e.g.: download_model.py 1
24M')
    sys.exit(1)
model = sys.argv[1]
subdir = os.path.join('models', model)
if not os.path.exists(subdir):
    os.makedirs(subdir)
subdir = subdir.replace('\', '/')
```

```

for filename in ['checkpoint', 'encoder.json', 'hparams.json', 'model.ckpt.data-00000-
of-00001', 'model.ckpt.index', 'model.ckpt.meta', 'vocab.bpe']:
    r = requests.get("https://openaipublic.blob.core.windows.net/gpt-
2/" + subdir + "/" + filename, stream=True)
    with open(os.path.join(subdir, filename), 'wb') as f:
        file_size = int(r.headers["content-length"])
        chunk_size = 1000
        with tqdm(ncols=100, desc="Fetching " + filename, total=file_size, unit_scale=
True) as pbar:
            for chunk in r.iter_content(chunk_size=chunk_size):
                f.write(chunk)
                pbar.update(chunk_size)

```

Файл encoder.py

```

import os
import json
import regex as re
from functools import lru_cache
@lru_cache()
def bytes_to_unicode():
    bs = list(range(ord("!"), ord("~")+1))+list(range(ord("¡"), ord("¬")+1))+list(range(
ord("®"), ord("ÿ")+1))
    cs = bs[:]
    n = 0
    for b in range(2**8):
        if b not in bs:
            bs.append(b)

```

```

cs.append(2**8+n)
    n += 1
cs = [chr(n) for n in cs]
return dict(zip(bs, cs))
def get_pairs(word):
    pairs = set()
    prev_char = word[0]
    for char in word[1:]:
        pairs.add((prev_char, char))
        prev_char = char
    return pairs
class Encoder:
    def __init__(self, encoder, bpe_merges, errors='replace'):
        self.encoder = encoder
        self.decoder = {v:k for k,v in self.encoder.items()}
        self.errors = errors # how to handle errors in decoding
        self.byte_encoder = bytes_to_unicode()
        self.byte_decoder = {v:k for k, v in self.byte_encoder.items()}
        self.bpe_ranks = dict(zip(bpe_merges, range(len(bpe_merges))))
        self.cache = {}
        self.pat = re.compile(r"""'s|'t|'re|'ve|'m|'ll|'d| ?\p{L}+| ?\p{N}+| ?[^\s\p{L}\p{N}]
]+|\s+(?!S)|\s+""")
    def bpe(self, token):
        if token in self.cache:
            return self.cache[token]
        word = tuple(token)
        pairs = get_pairs(word)
        if not pairs:

```

```
return token
while True:
    bigram = min(pairs, key = lambda pair: self.bpe_ranks.get(pair, float('inf')))
    if bigram not in self.bpe_ranks:
        break
    first, second = bigram
    new_word = []
    i = 0
    while i < len(word):
        try:
            j = word.index(first, i)
            new_word.extend(word[i:j])
            i = j
        except:
            new_word.extend(word[i:])
            break
        if word[i] == first and i < len(word)-1 and word[i+1] == second:
            new_word.append(first+second)
            i += 2
        else:
            new_word.append(word[i])
            i += 1
    new_word = tuple(new_word)
    word = new_word
    if len(word) == 1:
        break
    else:
        pairs = get_pairs(word)
```



```

word = ''.join(word)
    self.cache[token] = word
    return word
def encode(self, text):
    bpe_tokens = []
    for token in re.findall(self.pat, text):
        token = ''.join(self.byte_encoder[b] for b in token.encode('utf-8'))
        bpe_tokens.extend(self.encoder[bpe_token] for bpe_token in self.bpe(token).
split(' '))
    return bpe_tokens
def decode(self, tokens):
    text = ''.join([self.decoder[token] for token in tokens])
    text = bytearray([self.byte_decoder[c] for c in text]).decode('utf-
8', errors=self.errors)
    return text
def get_encoder(model_name, models_dir):
    with open(os.path.join(models_dir, model_name, 'encoder.json'), 'r') as f:
        encoder = json.load(f)
    with open(os.path.join(models_dir, model_name, 'vocab.bpe'), 'r', encoding="utf-
8") as f:
        bpe_data = f.read()
    bpe_merges = [tuple(merge_str.split()) for merge_str in bpe_data.split('\n')[1:-1]]
    return Encoder(
        encoder=encoder,
        bpe_merges=bpe_merges,
)

```

Файл GPT2-Learning.py

```
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters
import fire, json, os, string, sys, threading, random, model, sample, encoder, logging
, time
import numpy as np
import tensorflow as tf
import re
import os
debug = True
timstart = 1500
top = 0.66
temp = 1
mx = 0.00375
learning = ""
mode = False
learn = False
user = ""
cache = ""
running = False
def start(bot, update):
    global running
    global mode
    global learn
    global user
    global tim
    global learning
```

```

global cache
if user == "":
    user = update.message.from_user.id
    mode = True
    learn = True
    learning = ""
    cache = ""
    update.message.reply_text('I am GPT2 chatbot. Now I am in learning mode. Send me a message:')
elif user == update.message.from_user.id:
    mode = True
    learn = True
    learning = ""
    cache = ""
    update.message.reply_text('I am GPT2 chatbot. Now I am in learning mode. Send me a message:')
else:
    left = str(time)
    update.message.reply_text('Bot is currently in use, repeat after ' + left + ' seconds.')
def help(bot, update):
    update.message.reply_text('Just type a message... It could be lagged out. /chatbot goes into Me: You: mode. /finish just finishes the text /learn for conversation learning mode.')
def chatbot(bot, update):
    global running
    global mode
    global learn

```

```
global user
global tim
global learning
global cache
if user == "":
    user = update.message.from_user.id
    mode = True
    learn = False
    learning = ""
    cache = ""
    update.message.reply_text('Now I am in non-learning mode.')
elif user == update.message.from_user.id:
    mode = True
    learn = False
    learning = ""
    cache = ""
    update.message.reply_text('Now I am in non-learning mode.')
else:
    left = str(tim)
    update.message.reply_text('Bot is currently in use, repeat after ' + left + ' seconds.')
def finish(bot, update):
    global running
    global mode
    global learn
    global user
    global tim
    global learning
```

```

global cache
if user == "":
    user = update.message.from_user.id
    mode = False
    learn = False
    learning = ""
    cache = ""
    update.message.reply_text('Now I am in finishsentence mode.')
elif user == update.message.from_user.id:
    mode = False
    learn = False
    learning = ""
    cache = ""
    update.message.reply_text('Now I am in finishsentence mode.')
else:
    left = str(tim)
    update.message.reply_text('Bot is currently in use, repeat after ' + left + ' seconds.')
def learnon(bot, update):
    global running
    global mode
    global learn
    global user
    global tim
    global learning
    global cache
    if user == "":
        user = update.message.from_user.id

```

```

mode = True
learn = True
    learning = ""
    cache = ""
    update.message.reply_text('Now I am in learning mode.')
elif user == update.message.from_user.id:
    mode = True
    learn = True
    learning = ""
    cache = ""
    update.message.reply_text('Now I am in learning mode.')
else:
    left = str(tim)
    update.message.reply_text('Bot is currently in use, repeat after ' + left + ' seconds.')
def learnoff(bot, update):
    global running
    global mode
    global learn
    global user
    global tim
    global learning
    global cache
    if user == "":
        user = update.message.from_user.id
        mode = True
        learn = False
        learning = ""

```

```

cache = ""
update.message.reply_text('Now I am in non-learning mode.')
elif user == update.message.from_user.id:
    mode = True
    learn = False
    learning = ""
    cache = ""
    update.message.reply_text('Now I am in non-learning mode.')
else:
    left = str(tim)
    update.message.reply_text('Bot is currently in use, repeat after ' + left + ' seconds.')
def learnreset(bot, update):
    global running
    global mode
    global learn
    global user
    global tim
    global learning
    global cache
    if user == "":
        user = update.message.from_user.id
        mode = True
        learn = True
        learning = ""
        cache = ""
        update.message.reply_text('Now I am in learning mode.')
    elif user == update.message.from_user.id:

```

```

mode = True
learn = True
    learning = ""
    cache = ""
    update.message.reply_text('Now I am in learning mode.')
else:
    left = str(tim)
    update.message.reply_text('Bot is currently in use, repeat after ' + left + ' seconds.')

def regex(mew):
    mew = mew
    if "You:" in mew:
        mew = mew[0:mew.find('You:')]
    if "Me:" in mew:
        mew = mew[0:mew.find('Me:')]
    return mew
if "Me:" in mew:
    mew = mew[0:mew.find('Me:')]
    if "You:" in mew:
        mew = mew[0:mew.find('You:')]
    return mew
if "?" in mew:
    mew = mew[0:mew.find('?')]
    mew = mew + "?"
    return mew
if "!" in mew:
    mew = mew.rsplit('!', 1)[0]
    mew = mew + "!"

```



```

return meow
else:
    meow = meow.rsplit('.', 1)[0]
    meow = meow + "."
    return meow
meow = "Error."
return meow
def retry(bot, update):
    retr = True
    top_p = top
    temperature = temp
    mult = mx
    new = retr
    comput = threading.Thread(target=wait, args=(bot, update, top_p, temperature, m
ult, new,))
    comput.start()
def runn(bot, update):
    retr = False
    top_p = top
    temperature = temp
    mult = mx
    new = retr
    comput = threading.Thread(target=wait, args=(bot, update, top_p, temperature, m
ult, new,))
    comput.start()
def wait(bot, update, top_p, temperature, mult, new):
    global tim
    global user

```

```

global running
global mode
global learn
global learning
global cache
if user == "":
    user = update.message.from_user.id
if user == update.message.from_user.id:
    user = update.message.from_user.id
    tim = timstart
    compute = threading.Thread(target=interact_model, args=(bot, update, top_p, t
emperature, mult, new,))
    compute.start()
if running == False:
    while tim > 1:
        running = True
        time.sleep(1)
        tim = tim - 1
    if running == True:
        mode = False
        learn = False
        learning = ""
        cache = ""
        user = ""
        update.message.reply_text('Timer has run down, bot has been reset into th
e default mode.')
        running = False
else:

```

```

left = str(tim)
update.message.reply_text('Bot is in use, current cooldown is: ' + left + ' second
s.')
```

```
def interact_model(bot, update, top_p, temperature, mult, new):
```

```

    model_name = '355M'
    seed = random.randint(1431655765, 2863311530)
    nsamples = 1
    batch_size = 1
    top_k = 0
    models_dir = 'models'
    tex = update.message.text
    penguin = str(tex)
    global learning
    global learn
    global mode
    global cache
    if mode == True:
        cat = len(penguin.split())
        if cat > 300:
            update.message.reply_text('Input text is too long.')
            return
    if new == True and cache:
        m = re.search('.* You: ', cache)
        raw_text = m.group(0)
        cac = len(raw_text.split())
        cat = cac - 2
        length = cat
        if cat < 20:
```

```
length = 20
if cat > 20:
    length = 20
if cat > 30:
    length = 40
if cat > 50:
    length = 60
if debug == True:
    print("Cache is...")
    print(raw_text)
if new != True:
    wolf = 'Me: ' + penguin
    initial = wolf + ' You: '
    raw_text = learning + initial
    length = cat
if cat < 20:
    length = 20
if cat > 20:
    length = 20
if cat > 30:
    length = 40
if cat > 50:
    length = 60
    cache = raw_text
tgt = len(raw_text.split())
if tgt > 300:
    while tgt > 300:
        if debug == True:
```

```
print("Reducing memory of chat.")
    raw_text = raw_text.split(' Me:', 1)[-1]
    raw_text = "Me:" + raw_text
    tgt = len(raw_text.split())
    if tgt > 300:
        if debug == True:
            print("Reducing memory of chat.")
            raw_text = raw_text.split('You:', 1)[-1]
            raw_text = "You:" + raw_text
            tgt = len(raw_text.split())
        if debug == True:
            print("FINAL MEMORY REDUCTION:")
            print(raw_text)
if mode == False:
    cat = len(penguin.split())
    length = cat
    if length > 300:
        update.message.reply_text('Input text is too long.')
        return
    if new != True:
        cache = penguin
    if new == True and cache:
        penguin = cache
        length = len(penguin.split())
        cat = length
    if debug == True:
        print("Cache is...")
        print(penguin)
```

```

raw_text = penguin
tx = float(top_p)
cax = float(cat)
cay = float(mx)
caz = float(cax * cay)
top_p = caz + tx
if top_p > 0.84:
    top_p = 0.84
if top_p < 0.005:
    top_p = 0.005
models_dir = os.path.expanduser(os.path.expandvars(models_dir))
if batch_size is None:
    batch_size = 1
assert nsamples % batch_size == 0
enc = encoder.get_encoder(model_name, models_dir)
hparams = model.default_hparams()
with open(os.path.join(models_dir, model_name, 'hparams.json')) as f:
    hparams.override_from_dict(json.load(f))
if length is None:
    length = hparams.n_ctx // 2
elif length > hparams.n_ctx:
    raise ValueError("Can't get samples longer than window size: %s" % hparams.
n_ctx)
with tf.Session(graph=tf.Graph()) as sess:
    context = tf.placeholder(tf.int32, [batch_size, None])
    np.random.seed(seed)
    tf.set_random_seed(seed)
    output = sample.sample_sequence(

```

```

hparams=hparams, length=length,
    context=context,
    batch_size=batch_size,
    temperature=temperature, top_k=top_k, top_p=top_p
)
saver = tf.train.Saver()
ckpt = tf.train.latest_checkpoint(os.path.join(models_dir, model_name))
saver.restore(sess, ckpt)
context_tokens = enc.encode(raw_text)
generated = 0
for _ in range(nsamples // batch_size):
    out = sess.run(output, feed_dict={
        context: [context_tokens for _ in range(batch_size)]
    })[:, len(context_tokens):]
    for i in range(batch_size):
        generated += 1
        text = enc.decode(out[i])
        if debug == True:
            print("=====")
            print("Before splitlines: " + text)
            print("=====")
        if mode == True:
            pika = text.splitlines()[0]
        else:
            pika = text
        stripes = pika.encode(encoding=sys.stdout.encoding,errors='ignore')
        tigger = stripes.decode("utf-8")
        mew = str(tigger)

```

```
if mode == True:
    meo = regex(mew)
    meow = " ".join(re.split("[^a-zA-Z.,?!*]+", meo))
else:
    meow = mew
if learn == True:
    learning = raw_text + meow + " "
update.message.reply_text(meow)
if debug == True:
    print("=====")
    mod = str(mode)
    print("Mode: " + mod)
    lear = str(learn)
    print("Learn: " + lear)
    lent = str(length)
    print("Length: " + lent)
    print("=====")
    ball = str(pika)
    print("Before regex: " + ball)
    print("=====")
    print("Output: " + meow)
    print("=====")
    print("Raw_text or Original: " + raw_text)
    print("=====")
    print("Learning text or Next: " + learning)
    print("=====")
    tps = str(top_p)
    print("top_p out: " + tps)
```



```

print("=====")
    tpa = str(tx)
    print("top_p in: " + tpa)
    print("=====")
sess.close()
def error(bot, update):
    logger.warning("Update \"%s\" caused error \"%s\"", update)
def main():
    updater = Updater("1835075256:AAFk-U3ohhgHw6W-
efG00j9kDj8xMHXGSo8", use_context=False)
    dp = updater.dispatcher
    dp.add_handler(CommandHandler("start", start))
    dp.add_handler(CommandHandler("help", help))
    dp.add_handler(CommandHandler("chatbot", chatbot))
    dp.add_handler(CommandHandler("finish", finish))
    dp.add_handler(CommandHandler("learnon", learnon))
    dp.add_handler(CommandHandler("learnoff", learnoff))
    dp.add_handler(CommandHandler("learnreset", learnreset))
    dp.add_handler(CommandHandler("retry", retry))
    dp.add_handler(MessageHandler(Filters.text, runn))
    dp.add_error_handler(error)
    updater.start_polling()
    updater.idle()
if __name__ == '__main__':
    fire.Fire(main)

```

Файл model.py

```

import numpy as np
import tensorflow as tf
from tensorflow.contrib.training import HParams
def default_hparams():
    return HParams(
        n_vocab=0,
        n_ctx=1024,
        n_embd=768,
        n_head=12,
        n_layer=12,
    )
def shape_list(x):
    static = x.shape.as_list()
    dynamic = tf.shape(x)
    return [dynamic[i] if s is None else s for i, s in enumerate(static)]
def softmax(x, axis=-1):
    x = x - tf.reduce_max(x, axis=axis, keepdims=True)
    ex = tf.exp(x)
    return ex / tf.reduce_sum(ex, axis=axis, keepdims=True)
def gelu(x):
    return 0.5*x*(1+tf.tanh(np.sqrt(2/np.pi)*(x+0.044715*tf.pow(x, 3))))
def norm(x, scope, *, axis=-1, epsilon=1e-5):
    with tf.variable_scope(scope):
        n_state = x.shape[-1].value
        g = tf.get_variable('g', [n_state], initializer=tf.constant_initializer(1))
        b = tf.get_variable('b', [n_state], initializer=tf.constant_initializer(0))

```

```

    u = tf.reduce_mean(x, axis=axis, keepdims=True)
    s = tf.reduce_mean(tf.square(x-u), axis=axis, keepdims=True)
    x = (x - u) * tf.rsqrt(s + epsilon)
    x = x*g + b
    return x

def split_states(x, n):
    *start, m = shape_list(x)
    return tf.reshape(x, start + [n, m//n])

def merge_states(x):
    *start, a, b = shape_list(x)
    return tf.reshape(x, start + [a*b])

def conv1d(x, scope, nf, *, w_init_stdev=0.02):
    with tf.variable_scope(scope):
        *start, nx = shape_list(x)
        w = tf.get_variable('w', [1, nx, nf],
initializer=tf.random_normal_initializer(stddev=w_init_stdev))
        b = tf.get_variable('b', [nf], initializer=tf.constant_initializer(0))
        c = tf.reshape(tf.matmul(tf.reshape(x, [-1, nx]), tf.reshape(w, [-1, nf]))+b,
start+[nf])
        return c

def attention_mask(nd, ns, *, dtype):
    i = tf.range(nd)[:None]
    j = tf.range(ns)
    m = i >= j - ns + nd
    return tf.cast(m, dtype)

def attn(x, scope, n_state, *, past, hparams):
    assert x.shape.ndims == 3
    assert n_state % hparams.n_head == 0

```

```

if past is not None:
    assert past.shape.ndims == 5
def split_heads(x):
    return tf.transpose(split_states(x, hparams.n_head), [0, 2, 1, 3])
def merge_heads(x):
    return merge_states(tf.transpose(x, [0, 2, 1, 3]))
def mask_attn_weights(w):
    _, _, nd, ns = shape_list(w)
    b = attention_mask(nd, ns, dtype=w.dtype)
    b = tf.reshape(b, [1, 1, nd, ns])
    w = w*b - tf.cast(1e10, w.dtype)*(1-b)
    return w
def multihead_attn(q, k, v):
    w = tf.matmul(q, k, transpose_b=True)
    w = w * tf.rsqrt(tf.cast(v.shape[-1].value, w.dtype))
    w = mask_attn_weights(w)
    w = softmax(w)
    a = tf.matmul(w, v)
    return a
with tf.variable_scope(scope):
    c = conv1d(x, 'c_attn', n_state*3)
    q, k, v = map(split_heads, tf.split(c, 3, axis=2))
    present = tf.stack([k, v], axis=1)
    if past is not None:
        pk, pv = tf.unstack(past, axis=1)
        k = tf.concat([pk, k], axis=-2)
        v = tf.concat([pv, v], axis=-2)
    a = multihead_attn(q, k, v)

```

```

    a = merge_heads(a)
    a = conv1d(a, 'c_proj', n_state)
    return a, present
def mlp(x, scope, n_state, *, hparams):
    with tf.variable_scope(scope):
        nx = x.shape[-1].value
        h = gelu(conv1d(x, 'c_fc', n_state))
        h2 = conv1d(h, 'c_proj', nx)
        return h2
def block(x, scope, *, past, hparams):
    with tf.variable_scope(scope):
        nx = x.shape[-1].value
        a, present = attn(norm(x, 'ln_1'), 'attn', nx, past=past, hparams=hparams)
        x = x + a
        m = mlp(norm(x, 'ln_2'), 'mlp', nx*4, hparams=hparams)
        x = x + m
        return x, present
def past_shape(*, hparams, batch_size=None, sequence=None):
    return [batch_size, hparams.n_layer, 2, hparams.n_head, sequence,
hparams.n_embd // hparams.n_head]
def expand_tile(value, size):
    value = tf.convert_to_tensor(value, name='value')
    ndims = value.shape.ndims
    return tf.tile(tf.expand_dims(value, axis=0), [size] + [1]*ndims)
def positions_for(tokens, past_length):
    batch_size = tf.shape(tokens)[0]
    nsteps = tf.shape(tokens)[1]
    return expand_tile(past_length + tf.range(nsteps), batch_size)

```

```

def model(hparams, X, past=None, scope='model', reuse=False):
    with tf.variable_scope(scope, reuse=reuse):
        results = {}
        batch, sequence = shape_list(X)
        wpe = tf.get_variable('wpe', [hparams.n_ctx, hparams.n_embd],
                              initializer=tf.random_normal_initializer(stddev=0.01))
        wte = tf.get_variable('wte', [hparams.n_vocab, hparams.n_embd],
                              initializer=tf.random_normal_initializer(stddev=0.02))
        past_length = 0 if past is None else tf.shape(past)[-2]
        h = tf.gather(wte, X) + tf.gather(wpe, positions_for(X, past_length))
        presents = []
        pasts = tf.unstack(past, axis=1) if past is not None else [None] * hparams.n_layer
        assert len(pasts) == hparams.n_layer
        for layer, past in enumerate(pasts):
            h, present = block(h, 'h%d' % layer, past=past, hparams=hparams)
            presents.append(present)
        results['present'] = tf.stack(presents, axis=1)
        h = norm(h, 'ln_f')
        h_flat = tf.reshape(h, [batch*sequence, hparams.n_embd])
        logits = tf.matmul(h_flat, wte, transpose_b=True)
        logits = tf.reshape(logits, [batch, sequence, hparams.n_vocab])
        results['logits'] = logits
    return results

```

Файл sample.py

```

import tensorflow as tf
import model
def top_k_logits(logits, k):
    if k == 0:
        return logits
    def _top_k():
        values, _ = tf.nn.top_k(logits, k=k)
        min_values = values[:, -1, tf.newaxis]
        return tf.where(
            logits < min_values,
            tf.ones_like(logits, dtype=logits.dtype) * -1e10,
            logits,
        )
    return tf.cond(
        tf.equal(k, 0),
        lambda: logits,
        lambda: _top_k(),
    )
def top_p_logits(logits, p):
    batch, _ = logits.shape.as_list()
    sorted_logits = tf.sort(logits, direction='DESCENDING', axis=-1)
    cumulative_probs = tf.cumsum(tf.nn.softmax(sorted_logits, axis=-1), axis=-1)
    indices = tf.stack([
        tf.range(0, batch),
        tf.maximum(tf.reduce_sum(tf.cast(cumulative_probs <= p, tf.int32), axis=-1) -
1, 0),

```

```

], axis=-1)
min_values = tf.gather_nd(sorted_logits, indices)
return tf.where(
    logits < min_values,
    tf.ones_like(logits) * -1e10,
    logits,
)
def sample_sequence(*, hparams, length, start_token=None, batch_size=None,
context=None, temperature=1, top_k=0, top_p=1):
    if start_token is None:
        assert context is not None, 'Specify exactly one of start_token and context!'
    else:
        assert context is None, 'Specify exactly one of start_token and context!'
        context = tf.fill([batch_size, 1], start_token)
    def step(hparams, tokens, past=None):
        lm_output = model.model(hparams=hparams, X=tokens, past=past,
reuse=tf.AUTO_REUSE)
        logits = lm_output['logits'][:, :, :hparams.n_vocab]
        presents = lm_output['present']
        presents.set_shape(model.past_shape(hparams=hparams,
batch_size=batch_size))
        return {
            'logits': logits,
            'presents': presents,
        }
    with tf.name_scope('sample_sequence'):
        def body(past, prev, output):
            next_outputs = step(hparams, prev, past=past)

```



```

logits = next_outputs['logits'][:, -1, :] / tf.to_float(temperature)
logits = top_k_logits(logits, k=top_k)
logits = top_p_logits(logits, p=top_p)
samples = tf.multinomial(logits, num_samples=1, output_dtype=tf.int32)
return [
    next_outputs['presents'] if past is None else tf.concat([past,
next_outputs['presents']], axis=-2),
    samples,
    tf.concat([output, samples], axis=1) ]
past, prev, output = body(None, context, context)
def cond(*args):
    return True
_, _, tokens = tf.while_loop(
    cond=cond, body=body,
    maximum_iterations=length - 1,
    loop_vars=[
        past,
        prev,
        output
    ],
    shape_invariants=[
        tf.TensorShape(model.past_shape(hparams=hparams,
batch_size=batch_size)),
        tf.TensorShape([batch_size, None]),
        tf.TensorShape([batch_size, None]),
    ],
    back_prop=False, )
return tokens

```