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
____________Аліна Савченко

«______»____________2022 р.

ДИПЛОМНИЙ ПРОЕКТ

(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ БАКАЛАВРА

ЗА СПЕЦІАЛЬНІСТЮ

122 «КОМП’ЮТЕРНІ НАУКИ ТА ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ»

Тема: «Discord чат-бот на основі алгоритмів обробки природної мови та технологій data science»

Виконавець: Козловцев Дмитро Сергійович

Керівник: к.т.н. доц. Зудов Олег Миколайович

Нормоконтролер: доц. Гізун Андрій Іванович

КИЇВ 2022

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет *кібербезпеки, комп'ютерної та програмної інженерії*
Кафедра *Комп'ютерних інформаційних технологій*
Освітній ступінь *Бакалавр*
Спеціальність *122 «Комп'ютерні науки»*
Освітньо-професійна програма *«Інформаційні технології проектування»*

ЗАТВЕРДЖУЮ
Завідувач кафедри
_________Аліна Савченко
«_____»___________2022 р.

ЗАВДАННЯ
на виконання дипломного проекту

студента: *Козловцева Дмитра Сергійовича*

1. Тема дипломного проекту *«Discord чат-бот на основі алгоритмів обробки природної мови та технологій data science»*

затверджена наказом ректора від « 29» 04. 2022 р. № 454/ст

2. Термін виконання проекту: з 09.05.2022 р. по 19.06.2022 р.

3. Вихідні дані до проекту: 1. Вибрати алгоритм на основі аналізу існуючих моделей обробки природної мови. 2. Написати і натренувати модель. 3. Поставити бота на API діскорду. 4. Забезпечити безперервну можливість спілкування с ботов у реальному часі.

4. Зміст пояснювальної записки: підготовка теоретичного матеріалу по роботі з трансформерами та API діскорду, а також написання самого боту.

Розділ 1. ТЕОРІЯ АЛГОРИТМІВ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ
Розділ 2. ІНСТРУМЕНТИ РОЗРОБКИ БОТА ТА НЕЙРОННОЇ МЕРЕЖІ
Розділ 3. ПРОЕКТУВАННЯ ДІСКОРД-БОТУ

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

Слайди презентації засобами MS Power Point: 1. Актуальність та практична цінність. 2. Мета, об'єкт та предмет дослідження. 3. Структура РНМ. 4. Структура ЗНМ. 5. Структура трансформерів. 6. Опис предметної області. 7. Приклади роботи боту. 8.Фінальний вигляд. 9. Створення таблиць. 10. Вставка даних за допомогою оператора INSERT. 11. Вставка даних за допомогою оператора LOAD. 12. Вигляд таблиць. 13. Висновки.

6. Календарний план-графік

№ п/п	Етапи виконання дипломного проекту	Термін виконання етапів	Примітки
1	Аналіз літератури та джерел за темою дипломного проекту.	10.05.2022р. - 12.05.2022	
2	Розробка та затвердження плану дипломного проекту.	13.05.2022р.	
3	Проведення консультації з науковим керівником щодо створення першого розділу.	14.05.2022р.	
4	Аналітичний огляд і постановка задачі.	15.05.2022р. - 22.05.2022р.	
5	Порівняльний аналіз існуючих систем	19.05.2022р. - 22.05.2022р.	
6	Опис функціональних можливостей	23.05.2022р. - 27.05.2022р.	
7	Розробка застосунку	28.05.2022р. - 04.06.2022р.	
8	Висновки та оформлення пояснювальної записки дипломного проекту.	05.06.2022р. – 08.06.2022р.	
9	Підписання необхідних документів у встановленому порядку.	09.06.2022р. – 10.06.2022р.	
10	Підготовка до захисту та попередній захист дипломного проекту на випусковій кафедрі дипломного проекту.	11.06.2021р. – 12.06.2021р.	

7. Дата видачі завдання: травня 2022 р.

Керівник дипломного проекту ___________ Зудов Олег Миколайович

(підпис керівника)   (П.І.Б.)

Завдання прийняв до виконання __________ Козловцев Дмитро Сергійович

(підпис випускника)   (П.І.Б.)

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Discord чат-бот на основі алгоритмів обробки природної мови та технологій data science» 57 сторінок, 19 рисунків, 30 використаних джерела.

ШНМ, ОПМ, ЗНМ, ШІ, ЗПЧ, ДКП, ДІСКОРД-БОТ, DIALOGPT2, HUGGINGFACE, ГРАДІЄНТНИЙ СПУСК, ФУНКЦІЇ АКТИВАЦІЇ.

Об’єкт дослідження: діскорд-бот на основі трансформеру DialoGPT.

Мета: отримати кінцевий продукт проектування бота за допомогою алгоритма обробки природної мови і проаналізувати рівень розуміння та формулювання речень.

Мета розділена на декілька завдань: накопичення теоретичної інформації розробка штучної нейронної мережі на основі трансформеру DialoGPT, розробка діскорд-бота.

Для досягнення мети потрібно чітко побудувати теоретичну базу починаючи від розуміння роботи елементарного персептрону, закінчуючи комплексними трансформерами, які є нащадками змішання рекурентних та згорткових нейронних мереж; правильно налаштувати модель; закодувати дані у потрібний формат; знайти і використати підходящу середу розробки, яка надає можливість швидко опрацювати велику кількість розрахунків з ціллю тренування моделі; закодити діскорд-бота, використовуючи діскорд API. Для побудови і навчання моделі був використаний Google Colaboratory.

В результаті виконання дипломного проекту було побудовано діскорд-бота на основі трансформеру DialoGPT, який імітує поведінку Гаррі Поттера, базуючись на репліках персонажів з першої книги. Також було оцінено рівень розумності бота.

Перелік Прийнятих скорочень:

ШНМ - штучні нейронні мережі.

ОПМ - обробка природної мови.

ЗНМ – згорткова нейронна мережа.

ШІ – штучний інтелект.

ЗПЧ – зворотнє поширення в часі.

ДКП – довга короткочасна пам’ять.

Зміст

ВСТУП

На сьогоднішній день штучні нейронні мережі використовуються в багатьох галузях життя, від керування автомобілем (автопілот Tesla) до створення списку покупок за допомогою голосової команди (помічник Siri в айфон). Така популярність і масштабність застосування є наслідком багатьох років експериментів та досліджень. Те, що починалось як намір створити програму, яка може «вивчити» елементарні числові концепції, перетворилося у величезну масу областей, де кожна область відповідає за конкретну задачу, серед яких варто відмітити розпізнавання зображень, обробку природної мови, класифікацію, регресію і багато інших.

Застосування штучних нейронних мереж має великий потенціал. Навіть зараз ми маємо змогу використовувати подібні алгоритми в медицині при створенні нових ліків або знаходження ракових клітин на ранній стадії розвитку в організмі людини. Подібний потенціал потребує талановитих кадрів і захоплених цікавою темою людей. Гуманне, етичне і ефективне використання та проектування ШНМ повинне популізуватись.

З метою подальших досліджень і популізації ШНМ почали використовувати у сфері розваг. Не тільки алгоритми, що створюють списки рекомендованих фільмів та відео для перегляду, але й такі моделі, з якими навіть технічно не підкована людина може експереметувати і отримувати задоволення. Прикладом може слугувати AI Dungeon: гра, яка надає юзеру можливість переживати пригоди у світі динамічно створеному алгоритмом на основі спілкування з гравцем.

Не є виключенням і чат-боти, за допомогою яких користувач мессенджеру або соціальної мережі може веесело провести час, очікуючи на друзів. Подібні проєкти є важливими ще й тому, що надають можливість розробникам без досвіду почати працювати з комплексними моделями і методами розробки ШНМ.

РОЗДІЛ 1 ТЕОРІЯ АЛГОРИТМІВ ОБРОБКИ ПРИРОДНОЇ МОВИ ТА ШТУЧНИХ НЕЙРОННИХ МЕРЕЖ

1.1 Штучні нейронні мережі і глибинне навчання

Глибинне навчання - це концепт, який бере своє натхнення з функцій людського мозку для удосконалення вирішення задач за допомогою машинного навчання. Концепт включає в себе алгоритми нейронної мережі, які намагаються імітувати роботу людського мозку та вчитися на отриманому досвіді.

Нейронні мережі - це обчислювальні алгоритми, котрі використовують мережу функцій для опрацювання та перетворення введених даних в бажаний результат.

Тому можемо зробити висновок, що нейронні мережі - це пошук закономірностей та зв'язків в даних за допомогою певних алгоритмів.

ШНМ має певну структуру:

- Вхідний шар - це вхідні дані, які потім потрапляють для опрацювання до прихованих шарів моделі.
- Прихований шар - це набір деякої кількості нейронів або вузлів, де виконуються усі обчислення та перетворення даних. Найпростіша мережа складається з одного прихованого шару, але в основному модель складається з декількох шарів задля покращення її ефективності.
- Вихідний шар - це отриманий в процесі обробки прихованим шаром результат. У вихідному шарі може бути як одна так і багато нод.

Кафедра КІТ				НАУ 22 08 20 − 000 ПЗ			
Виконав	*Козловцев Д.С.*			Discord чат-бот на основі алгоритмів обробки природної мови та технологій data science	Літ.	Арк.	Аркушів
Керівник	*Зудов О.М.*					7	21
Консульт.					122 ТП-415Б		
Нормоконт	*Гізун А.І.*						

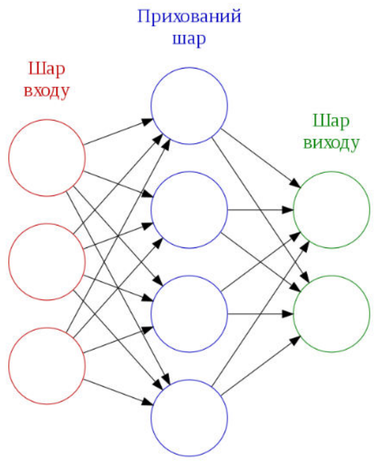


Рис.1.1 Структура ШНМ

[1-5]

1.2 Перцептрон і багатошаровий персептрон

Перцептрон є найпростішим видом нейронної мережі, який складається лише з одного шару, на якому виконуються всі математичні перетворення.

В тей час як багатошаровий перцептрон або ж штучна нейронна мережа, складається з більш ніж одного шару.

На рисунку зображено багатошаровий перцептрон який складається з чотирьох шарів, 2 з котрих є прихованими шарами.

- Вхідний шар із 6 вхідними вузлами.
- Прихований шар 1, з 4 прихованими вузлами/4 персептронами.
- Прихований шар 2, з 4 прихованими вузлами.
- Вихідний шар з 1 вихідним вузлом.

Покроковий алгоритм роботи багатошарового перцептрону:

1. На першому кроці передаються вхідні дані x1,x2, … xn з деякими ваговими коефіцієнтами w1,w1, … wn до прихованого шару.

2. Кожен прихований шар складається з нейронів або вузлів. Усі входи підключені до кожного вузла.

3. Всі обчислення виконуються у вузлі прихованого шару (синій овал на рисунку)

Обчислення виконуються в два етапи[6]:

Перш за все, всі вхідні дані множаться на відповідні їм вагові коефіцєнти. Коефіцієнти показують силу конкретного значення. Після призначення коефіцієнтів додається змінна зміщення.

Зміщення - це константа, яка допомагає моделі якнайкраще відповідати дійсності.

Z1 = W1*In1 + W2*In2 + W3*In3 + W4*In4 + W5*In5 + b

W1, W2, W3, W4, W5 - це коефіцієнти, призначені для вхідних даних In1, In2, In3, In4, In5, та b - зміщення.

Наступним кроком функція активації застосовується до лінійного рівняння Z1.

Функція активації - це нелінійне перетворення, яке застосовується до входу перед відправкою його до наступного шару нейронів. Важливість функції активації полягає у впровадженні нелінійності в модель.

4. Весь процес, описаний у пункті 3, виконується в кожному прихованому шарі. Після проходження через кожен прихований шар алгоритм переходить до останнього, або ж вихідного шару, який надає кінцевий результат.

Описані зверху чотири пункти відтворюють процес, який називається **пряме поширення**.

5. Після отримання прогнозів із вихідного шару обчислюється помилка, тобто різниця між дійсним і прогнозованим результатом. Якщо помилка велика, то вживаються заходи для мінімізації помилки, і з цією ж метою виконується зворотне поширення.

Зворотне поширення - це процес оновлення та пошуку оптимальних значень вагових коефіцієнтів, який допомагає моделі мінімізувати помилку.

Вагові показники оновлюються за допомогою оптимізаторів. Оптимізатори – це методи для зміни вагових коефіцієнтів нейронних мереж.

1.3 Градієнтний спуск

Градієнтний спуск є одним із поширених оптимізаторів, який допомагає підрахувати нові ваги.

На зображенні нижче зображена крива функції витрат, а мета - мінімізувати помилку, щоб досягнути Jмін, тобто глобального мінімуму.

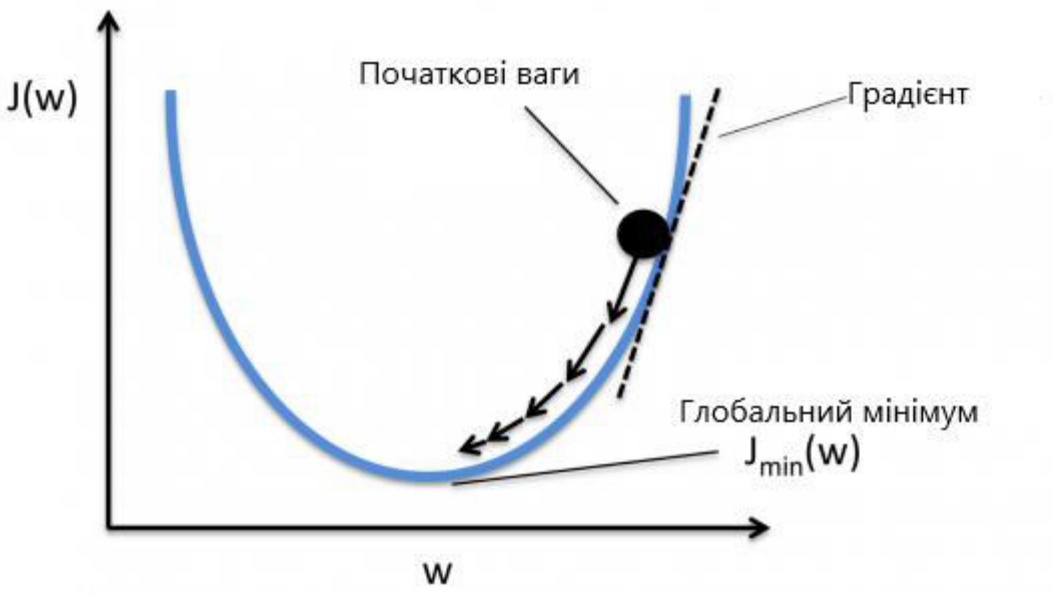


Рис1.2 Градієнтний спуск

Кроки для досягнення глобального мінімуму:

- Спочатку вагові коефіцієнти визначаються випадковим чином, тобто випадкове значення ваги і кутових коефіцієнтів призначаються моделі під час прямого поширення, а помилки вираховується після всіх обчислень
- Розраховується градієнт, тобто похідна похибки від поточних ваг
- Нові ваги розраховуються за наведеною нижче формулою, де а - швидкість навчання, яка також відома як розмір кроку для керування швидкістю або кроками зворотного поширення. Це дає додатковий контроль над тим, як швидко ми хочемо рухатися по кривій, щоб досягти глобальний мінімум.
- Цей процес обчислення нових ваг та помилок з нових ваг та оновлення ваг продовжується, поки ми не досягнемо глобального мінімуму, а похибки не будуть зведені до мінімуму.

Тут слід зазначити, що швидкість навчання в рівнянні для оновлення ваги, слід вибирати з обережністю.

Швидкість навчання – це кількість змін або розмір кроку, зроблений для досягнення глобального мінімуму. Він не повинен бути замалим, оскільки потрібен час для зближення, але також не повинен бути завеликим, щоб він взагалі не досягав глобального мінімуму. Отже, швидкість навчання є гіперпараметром, який ми повинні вибрати на основі моделі.

[4-8]

1.4 Функції активації

Функції активації приєднані до кожного нейрона і є математичними рівняннями, які визначають активацію нейрона на основі впливу входу нейрона на передбачення моделі. Головною метою функції активації є введення нелінійності в дані[5].

Різні типи функцій активації:

- Сигмовидна активаційна функція
- Tanh- функція активації гіперболічного тангенса
- Випрямлена лінійна одинична функція (ReLU)
- Нещільний ReLU
- Softmax

1.5 Типи нейронних мереж

Нейронні мережі іноді описують з точки зору їх глибини, включаючи те, скільки шарів вони мають між введенням і виведенням, або так званих прихованих шарів моделі. Ось чому термін нейронна мережа використовується майже як синонім глибокого навчання.

Вони можуть бути описані як кількістю прихованих вузлів які має модель, так і кількістю входів та виходів у кожного вузла. Варіації класичного дизайну нейронної мережі дозволяють різні форми прямого і зворотного поширення інформації між рівнями.

Конкретні типи ШНМ:

• Нейронні мережі з прямим зв'язком: один з найпростіших варіантів нейронних мереж. Вони передають інформацію в одному напрямку, через різні вхідні вузли, поки вона не потрапить до вихідного вузла. Мережа може мати або не мати прихованих шарів вузлів, що робить їх функціонування більш інтерпретованим. Він готовий до обробки великої кількості шуму. Цей тип обчислювальної моделі використовується в класифікації та регресії.

- Рекуррентні нейронні мережі складніші. Вони зберігають вихідні дані вузлів обробки та повертають результат до моделі. Кожен вузол у моделі діє як комірка пам'яті, продовжуючи обчислення та виконання операцій. Ця нейронна мережа починається з того ж прямого поширення, що й мережа прямого зв'язку, але потім продовжує запам'ятовувати всю оброблену інформацію, щоб повторно використовувати її в майбутньому. Якщо прогноз мережі неправильний, то система самонавчається і продовжує працювати над правильним прогнозом під час зворотного поширення. Цей тип часто використовується для перетворення тексту в мову.
- Згорткові нейронні мережі: одна з найпопулярніших моделей, що використовуються сьогодні. Ця обчислювальна модель нейронної мережі використовує різновид багатошарових перцептронів і містить один або кілька згорткових шарів, які можуть бути повністю пов'язані або об'єднані в пул. Ці згорткові шари створюють карти властивостей, які записують область зображення, яке в кінцевому підсумку розбивається на прямокутники і надсилається для нелінійного моделювання. Модель особливо популярна у сфері розпізнавання зображень; його використовували в багатьох найсучасніших застосувань ШІ, включаючи розпізнавання обличчя, оцифрування тексту та обробка природної мови. Інші види використання включають виявлення парафраз, обробку сигналу та класифікацію зображень.
- Деконволюційні нейронні мережі: використовуйте процес зворотної моделі ЗНМ. Вони прагнуть знайти втрачені функції або сигнали, які спочатку вважалися неважливими для завдання системи згорткової нейронної мережі. Ця мережева модель може бути використана для синтезу та аналізу зображень.
- Модульні нейронні мережі: містять кілька нейронних мереж, що працюють окремо одна від одної.

[4-5]

1.6 Переваги і недоліки нейронних мереж

До переваг штучних нейронних мереж можна віднести:

- Можливості паралельної обробки. Мережа може виконувати більше однієї роботи одночасно.
- Інформація зберігається у всій мережі, а не тільки в базі даних.
- Здатність навчатися та моделювати нелінійні, складні відносини допомагає моделювати реальні життєві відносини між входом і виходом.
- Відмовостійкість означає, що пошкодження однієї чи кількох осередків ШНМ не зупинить генерацію вихідних даних.
- Поступове пошкодження означає, що мережа буде повільно деградувати з часом, замість того, щоб проблема миттєво зруйнувала мережу.
- Здатність виробляти результати з неповними знаннями з втратою продуктивності залежить від того, наскільки важливою є відсутня інформація.
- На вхідні змінні, наприклад, на те, як вони мають розподілятися, не накладається жодних обмежень.
- Машинне навчання означає, що ШНМ може вчитися на подіях і приймати рішення на основі спостережень.
- Можливість вивчати приховані зв’язки в даних без керування будь-якими фіксованими зв’язками означає, що ШНМ може краще моделювати дуже мінливі дані та непостійну дисперсію.
- Здатність узагальнювати та виводити невидимі зв’язки з невидимими даними означає, що ШНМ можуть передбачати вихід невидимих даних.

Недоліки:

- Відсутність правил для визначення належної структури мережі означає, що відповідну архітектуру штучної нейронної мережі можна знайти лише шляхом проб, помилок, і досвіду.
- Вимоги до процесорів із можливостями паралельної обробки роблять нейронні мережі апаратно залежними.

- Мережа працює з числовою інформацією, тому всі проблеми повинні бути переведені в числові значення, перш ніж вони можуть бути представлені в ШНМ.
- Відсутність пояснень за рішеннями для зондування є одним із найбільших недоліків ШНМ. Неможливість пояснити, чому і як стоїть рішення, породжує відсутність довіри до мережі.

[4-5]

1.7 Застосування ШНМ

Розпізнавання зображень було однією з перших областей, до яких нейронні мережі були успішно застосовані, але використання технології поширилося на багато інших областей, включаючи:

- Чат-боти.
- Обробку природної мови, переклад і генерацію мови.
- Прогнози фондового ринку.
- Планування та оптимізація маршруту водіїв доставки.
- Відкриття та розробка ліків.

Це лише кілька конкретних областей, до яких сьогодні застосовуються нейронні мережі. Основне використання передбачає будь-який процес, який працює за суворими правилами або шаблонами і має велику кількість даних. Якщо дані занадто великі для людини, щоб зрозуміти їх за маленький проміжок часу, то цей процес, ймовірно, є головним кандидатом для автоматизації за допомогою штучних нейронних мереж.

1.8 Рекурентні нейронні мережі

Рекурентні нейронні мережі - це клас нейронних мереж, які допомагають моделювати послідовні дані. Похідні від мереж прямого зв’язку, вони демонструють поведінку подібну до того, як функціонує людський мозок.

Простіше кажучи: рекурентні нейронні мережі дають прогноз в послідовних даних, з якими, що важливо, не можуть або неефективно працюють інші алгоритми.

Послідовні дані - це просто впорядковані дані, у яких зв'язана контекстом інформація йде одна за одною.

Найпопулярнішим типом послідовних даних є дані часові ряди, які представляють собою серію точок даних, перерахованих у часовому порядку.

У РНМ інформація проходить по циклу. Коли алгоритм приймає рішення враховуються поточні дані, а також те, що він дізнався з введених даних, отриманих раніше, що додає концепт короткочасної пам'яті до алгоритму. Два зображення нижче ілюструють різницю в потоці інформації між РНМ і нейронною мережею з прямим зв’язком.

Звичайний РНМ має короткочасну пам'ять. За допомогою концепту довготривалої короткочасної пам'ятті (ДКП) цей вид ШНМ отримує можливість запам'ятовувати все речення.

Отже, РНМ має два вузли серед вхідних даних: теперішнє (дані з датасету) і недавнє минуле (прогноз минулої ітерації). Це важливо, оскільки послідовність даних містить інформацію про те, що буде далі, саме тому РНМ може робити операції, які інші алгоритми не можуть. Нейронна мережа з прямим зв’язком призначає, як і всі інші алгоритми глибокого навчання, вагову матрицю своїм входам, а потім виробляє вихідні дані, без запам'ятовування останніх.

Потрібно звернути увагу, що РНМ застосовують ваги до поточного, а також до попереднього входу. Крім того, рекурентна нейронна мережа також налаштовує ваги як для градієнтного спуску, так і для зворотного поширення в часі.

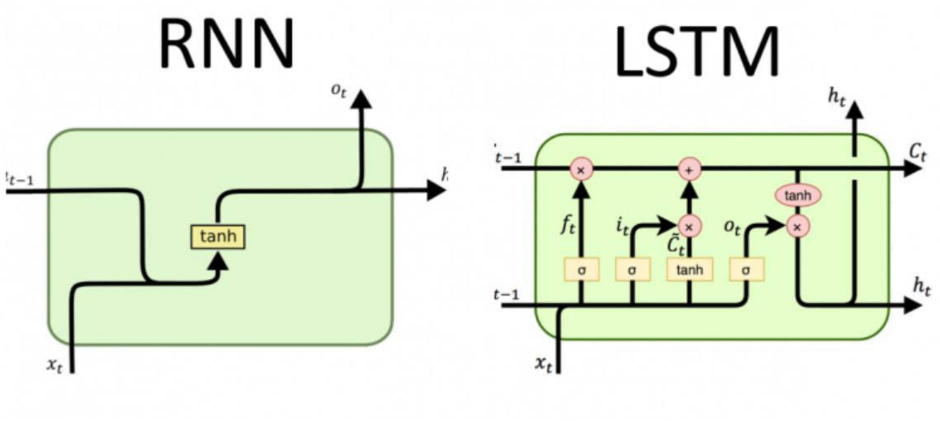


Рис.1.3 Архітектура РНМ

Види РНМ моделей

- Один до одного
- Один до багатьох
- Багато до одного
- Багато до багатьох

[9-11]

1.9 Зворотне поширення в часі

Зворотнє поширення з часом - це просто сучасне слово для зворотного поширення на розгорнутій РНМ[10]. Розгортання - це інструмент візуалізації та концептуальний інструмент, який допомагає зрозуміти, що відбувається в мережі. У більшості випадків, коли рекурентну нейронну мережу реалізують у звичайних фреймворках програмування, зворотне поширення відбувається автоматично, але потрібно розуміти як це працює, щоб усунути проблеми, які можуть виникнути в процесі розробки.

Можна розглядати РНМ як послідовність нейронних мереж, які навчаються одна за одною із зворотним поширенням[11].

У ЗПЧ помилка поширюється назад від останнього до першого часового кроку, одночасно розгортаючи всі часові кроки. Це дозволяє обчислювати похибку для кожного кроку часу, що в свою чергу надає можливість оновлювати вагові коефіцієнти.

Потрібно зауважити, що ЗПЧ може бути дорогим з точки зору обчислень, особливо якщо є велика кількість часових кроків.

1.10 Головні проблеми РНМ

- Вибухаючий градієнт- алгоритм без особливих причин надає високу важливість ваг. На щастя, цю проблему можна легко вирішити, обрізаючи градієнти або правильно конфігуруючи вагу нейронів під час проектування моделі.

- Зникаючий градієнт- відбувається, коли значення градієнта занадто малі і модель перестає навчатися або, в результаті маленьких значень ваг, займає занадто багато часу. Це була серйозна проблема 1990-х років, і її набагато важче вирішити, ніж вибухові градієнти. На щастя, це було вирішено за допомогою концепції довгої короткочасної пам’яті Сеппа Хохрайтера та Юргена Шмідхубера.

1.11 Довга короткосрочна пам'ять (ДКП)

Мережі довгострокової пам’яті (ДКП) є розширенням для рекурентних нейронних мереж. Таке розширення добре підходить для того, щоб вчитися на важливому досвіді який подається з великими часовми проміжками[11].

Одиниці ДКП використовуються як будівельні одиниці для шарів РНМ, які часто називають мережею ДКП.

ДКП дозволяють РНМ запам’ятовувати вхідні дані протягом тривалого періоду часу. Це можливо тому, що ДКП містять інформацію в пам’яті так само як пам’ять комп’ютера. ДКП може читати, записувати та видаляти інформацію зі своєї пам'яті.

Цю пам’ять можна розглядати клітинку, де вирішується, зберігається чи видаляється інформація, іншими словами,вирішується відкривати так звані ворота для даних чи ні. Рішення про відкриття/закриття воріт приймається алгоритмом на основі важливості інформація якою оперує узел в данний момент часу. Призначення важливості відбувається за допомогою ваг, які також вивчаються алгоритмом.

Це просто означає, що з часом модель дізнається, яка інформація важлива, а яка ні.

У ДКП є троє воріт: вхідний, забування і вихідний. Ці ворота визначають дію: дозволяти новий вхід (вхідні ворота), видаляти інформацію, оскільки вона не важлива (ворота забування), або дозволити даним вплинути на вихід в поточному часовому етапі (вихідні ворота). Нижче наведено ілюстрацію РНМ з трьома воротами:

Ворота в ДКП є аналоговими у формі сигмоїдів, тобто вони коливаються від нуля до одиниці. Той факт, що вони є аналоговими, дозволяє їм здійснювати зворотне поширення.

Проблемні питання зникнення градієнтів вирішуються за допомогою ДКП, оскільки він створює крутимі градієнти, що робить навчання відносно коротким, а точність - високою.

1.12 Згорткові нейронні мережі

Згорткова нейронна мережа - це певний вид нейронної мережі з кількома шарами. Він обробляє дані, які створені у виді сітки, а потім виводить параметри, які описують концепт за даними, що в свою чергу дає можливість визначити вивід. Однією з величезних переваг використання ЗНМ є те, що не потрібно робити багато попередньої обробки зображень.

У більшості алгоритмів, які обробляють зображення, фільтри зазвичай створюються інженером на основі еврістики. ЗНМ створюють фільтри самі і можуть дізнатися, які характеристики у фільтрах є найважливішими. Це заощаджує багато часу та роботи. Основна мета алгоритму згорткової нейронної мережі - передати дані у форми, які легше обробляти, не втрачаючи при цьому функцій, важливих для з’ясування того, що представляють дані. Це також робить їх чудовими кандидатами для обробки величезних наборів даних.

Велика різниця між ЗНМ і звичайною нейронною мережею полягає в тому, що ЗНМ використовують згортки для обробки математики за параметрами. Згортка використовується замість множення матриці принаймні в одному шарі ЗНМ.

ЗНМ працюють, застосовуючи фільтри до вхідних даних. Що робить їх такими особливими, так це те, що ЗНМ можуть налаштовувати фільтри під час навчання. Таким чином результати точно налаштовуються в режимі реального часу, навіть якщо набори даних величезні, як-от зображення.

Оскільки фільтри можна оновити, щоб краще тренувати ЗНМ, це усуває потребу в створених вручну фільтрів.

Це в свою чергу дає більшу гнучкість щодо кількості фільтрів, які ми можемо застосувати до набору даних. Використовуючи цей алгоритм, можливо працювати над більш складними проблемами, такими як розпізнавання обличчя.

Хоча мережі можна навчати з відносно невеликою кількістю даних (~10 000), чим більше даних буде доступно, тим краще буде налаштований ЗНМ.

Згорткові нейронні мережі засновані на висновках нейронауки. Вони складаються з шарів штучних нейронів. Вузли ЗНМ є функціями, які обчислюють зважену суму вхідних даних і повертають карту активації. Це частина згортки нейронної мережі.

Кожен вузол шару визначається своїми значеннями ваги. Коли моделі надаються дані, як-от зображення, шар приймає значення пікселів і виділяє деякі візуальні характеристики.

Працюючи з даними ЗНМ повертає карти активації на кожному рівні. Ці карти вказують на важливі особливості в наборі даних. Якщо надати ЗНМ зображення, то алгоритм вкаже функції на основі значень пікселів, як-от кольори, і надасть функцію активації.

Зазвичай із зображеннями ЗНМ спочатку знайде краї зображення, тоді це невелике визначення зображення буде передано до наступного шару, потім цей шар почне виявляти такі речі, як кути та кольорові групи. Це визначення зображення буде передано до наступного шару, і цикл продовжуватимеся, доки не буде зроблено передбачення.

Коли шари стають більш визначеними, в дію входить максимальне об’єднання. Завдяки йому алгоритм повертає лише найбільш релевантні функції з шару на карті активації. Саме ця інформація передається кожному наступному шару, поки не вийде останній шар.

Останній рівень ЗНМ - це рівень класифікації, який визначає прогнозоване значення на основі карти активації. Якщо передавати зразок рукописного вводу в ЗНМ, рівень класифікації скаже, яка буква/цифра на зображенні. Це те, що використовують автономні транспортні засоби, щоб визначити, чи є об’єкт іншим автомобілем, людиною або якоюсь іншою перешкодою.

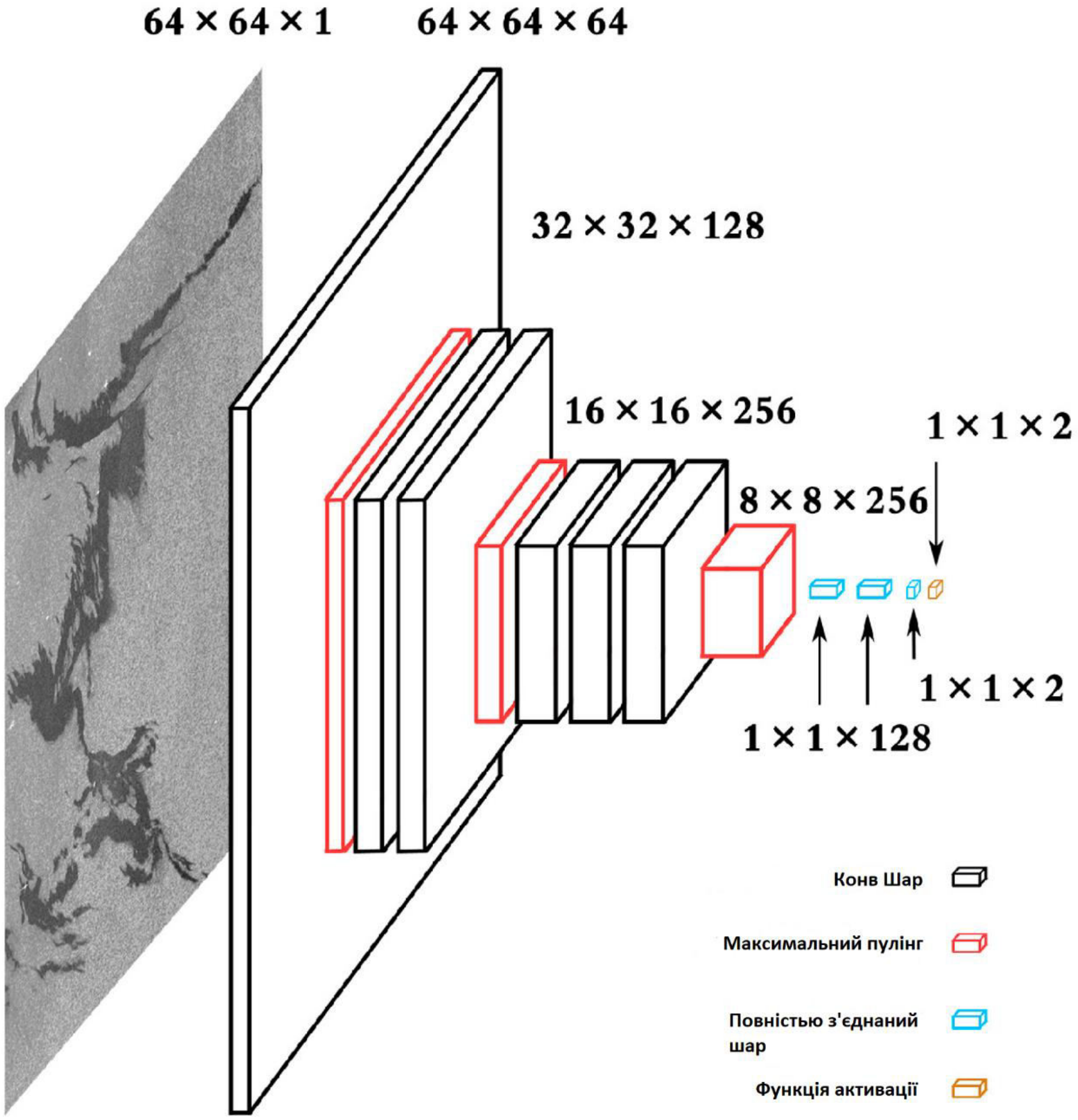


Рис.1.4 Архітектура ЗНМ

[12-13]

1.13 Трансформери

Архітектура трансформатора має кілька цікавих властивостей, які роблять його основою базових моделей:

Моделі-трансформери легко паралелізуються як для навчання, так і для висновків, оскільки, на відміну від попередньої хвилі сучасних моделей ОПМ, повторювані нейронні мережі (зазвичай у формі ДКП або ЗПБ), які покладаються на послідовні попередніх маркерів. У кожному шарі трансформатора обчислення відбуваються паралельно для всіх вхідних маркерів.

Все ще існує залежність від попередніх рівнів нейронної мережі, але майже кожна сучасна мережева архітектура має цю властивість.

Існує цікавий напрямок досліджень, зосереджений на розв’язанні обчислень уздовж кількох осей, і успіхи тут можуть змінити статус-кво.

Ще є деякі місця, де трансформатори стають послідовними, особливо у випадку авторегресивної генерації, або з різними модифікаціями оригінальної архітектури трансформерів, які певним чином включають повторення в трансформерах.

Ця паралелізація має свої обмеження, тому що вам потрібно багато пам'яті для зберігання всіх проміжних результатів, а самі обчислення масштабуються в квадратичному порядку з кількістю вхідних елементів у оригінальній архітектурі трансформатора. На практиці кодер обмежений максимальним розміром введення (зазвичай його називають «діапазоном уваги»), який знаходиться в діапазоні від 512 (для BERT) до 2048 (для GPT-3) маркерів. Рекурентні нейронні мережі не мають таких обмежень, але у них є інші проблеми (вибух/зникнення градієнта), що обмежують їх застосування до дійсно довгих послідовностей.

Існує окремий напрямок досліджень, спрямований на ефективні трансформери для довгих послідовностей, і подальші успіхи зроблять трансформери ще більш привабливими.

Архітектура трансформера має менше неявних зміщень, ніж, скажімо, згорткові (ЗНМ) або рекурентні (РНМ) нейронні мережі.

Неявні зміщення - це рішення щодо дизайну, які передбачають певну властивість у вхідних даних. Менше неявних зміщень з одного боку ускладнює навчання (це вимагає більше навчальних прикладів, щоб дізнатися щось корисне), але робить його більш універсальним.

Це означає, що можна надати більше даних, і модель сама засвоїть правильні параметри, і, можливо, ці параметри будуть кращими та більш підходящими для даної проблеми, ніж будь-які параметри, підібрані вручну. Ця історія нагадує перехід між класичним машинним навчанням і глибоким навчанням.

Моделі-трансформери можна згрупувати в три категорії:

- Подібні до GPT (також називаються авторегресивними моделями трансформера)
- Подібні до BERT (також називаються моделями трансформерами з автокодуванням)
- Подібні до BART/T5 (також називаються моделями трансформерами послідовності до послідовності)

Усі згадані вище моделі трансформерів (GPT, BERT, BART, T5 тощо) були навчені як мовні моделі. Це означає, що вони пройшли навчання на великій кількості необробленого тексту під самоконтролем. Самоконтрольне навчання – це тип навчання, в якому ціль автоматично обчислюється на основі вхідних даних моделі. Це означає, що людям не потрібно маркувати дані!

Цей тип моделі розвиває статистичне розуміння мови, на якій її навчали, але вона не дуже корисна для конкретних практичних завдань. Через це, загальна попередньо підготовлена модель потім проходить процес, який називається трансферним навчанням. Під час цього процесу модель налаштовується під наглядом, тобто з використанням анотованих людьми міток — для певного завдання.

Прикладом завдання є передбачення наступного слова в реченні, прочитавши n-кількість попередніх слів.

Це називається причинно-наслідковим мовним моделюванням, оскільки результат залежить від минулих і теперішніх вхідних даних, але не від майбутніх. Іншим прикладом є моделювання замаскованої мови, при якому модель передбачає замасковане слово в реченні.

Окрім кількох виключень (наприклад, DistilBERT), загальна стратегія досягнення кращої продуктивності полягає в збільшенні розмірів моделей, а також обсягу даних, на яких вони попередньо навчаються. На жаль, навчання моделі, особливо великої, вимагає великої кількості даних. Це стає дуже витратним з точки зору часу та обчислювальних ресурсів. Це навіть означає негативний вплив на навколишнє середовище. І ось чому загальний доступ до мовних моделей має першорядне значення: спільне використання навчених ваг і створення на основі вже навчених ваг зменшує загальні витрати на обчислення та вуглецевий слід спільноти.

[14-17]

1.15 Трансферне навчання

Попереднє навчання - це процес навчання моделі з нуля: ваги задаються випадковим чином, і навчання починається без будь-яких попередніх знань.

Це попереднє навчання зазвичай проводиться на дуже великих обсягах даних. Тому для цього потрібен дуже великий масив, а навчання може тривати до кількох тижнів.

Точне налаштування - це навчання, яке проводиться після попереднього навчання моделі. Щоб виконати точне налаштування, спочатку отримується попередньо навчена мовна модель, потім виконується додаткове навчання з набором даних, специфічним для вашого завдання. Модель не тренують безпосередньо для останнього завдання по декільком причинам:

- Попередньо навчену модель вже навчали на наборі даних, який має певну схожість з набором даних точного налаштування. Таким чином, процес налаштування може використовувати знання, отримані початковою моделлю під час попереднього навчання (наприклад, з проблемами НЛП, попередньо навчена модель матиме якесь статистичне розуміння мови).

- Оскільки попередньо тренована модель вже була навчена на великій кількості даних, точне налаштування вимагає набагато менше даних, щоб отримати гідні результати.
- З цієї ж причини кількість часу та ресурсів, необхідних для отримання хороших результатів, набагато менші.

Наприклад, можна використовувати попередньо підготовлену модель, навчену англійською мовою, а потім точно налаштувати її на корпусі arXiv, в результаті чого буде створена науково-дослідницька модель. Для точного налаштування буде потрібно лише обмежена кількість даних: знання, отримані попередньо підготовленою моделлю «передаються», звідси й термін «навчання з перенесенням».

1.16 Архітектура трансформера

Модель в основному складається з двох блоків:

- Кодер (ліворуч): кодер отримує вхідні дані та створює його представлення (його особливості). Це означає, що модель оптимізована для отримання розуміння з вхідних даних.
- Декодер (праворуч): декодер використовує представлення (функції) кодера разом з іншими вхідними сигналами для створення цільової послідовності. Це означає, що модель оптимізована для генерування вихідних даних.

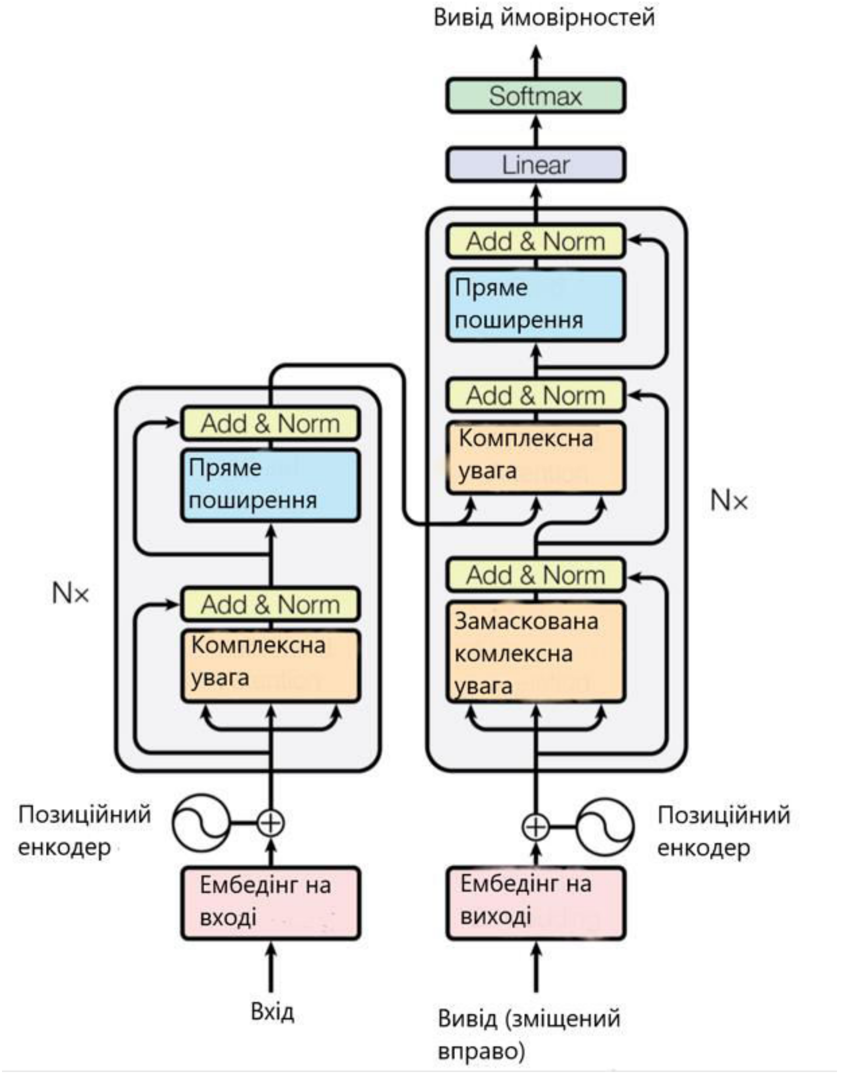


Рис.1.5 Архітектура трансформера

Ключовою особливістю моделей трансформерів є те, що вони складаються зі спеціальних шарів, які називаються шарами уваги. Цей шар скаже моделі звертати особливу увагу на певні слова в реченні, які йому передали (і більш-менш ігнорувати інші), коли має справу з представленням кожного слова.

Щоб помістити це в контекст, потрібно розглянути завдання перекладу тексту з англійської на французьку. Враховуючи вхідні дані «Вам подобається цей курс», моделі перекладу також потрібно буде враховувати сусіднє слово «You», щоб отримати правильний переклад для слова «like», оскільки у французькій мові дієслово «like» відмінюється по-різному залежно від предмета. Однак решта речення не є корисною для перекладу цього слова. У тому ж ключі при перекладі «це» моделі також потрібно буде звернути увагу на слово «курс», оскільки «це» перекладається по-різному залежно від того, чи пов’язаний іменник чоловічого або жіночого роду. Знову інші слова в реченні не матимуть значення для перекладу «це». З більш складними реченнями (і складнішими граматичними правилами) моделі потрібно буде приділяти особливу увагу словам, які можуть з’являтися далі в реченні, щоб правильно перекладати кожне слово.

Те саме поняття стосується будь-якого завдання, пов’язаного з природною мовою: слово саме по собі має значення, але на це значення глибоко впливає контекст, яке може бути будь-яким іншим словом (або словами) до або після слова, що вивчається.

Архітектура трансформер спочатку була розроблена для перекладу, що логічно, оскільки, як зазначалось вище, під час навчання кодер отримує вхідні дані (речення) певною мовою, а декодер отримує ті самі речення на бажаній цільовій мові. У кодері шари уваги можуть використовувати всі слова в реченні (оскільки, як ми щойно бачили, переклад даного слова може залежати від того, що стоїть після, а також перед ним у реченні). Однак декодер працює послідовно і може звертати увагу лише на слова в реченні, які він уже переклав (тому лише слова перед словом, яке в даний момент генерується). Наприклад, коли ми передбачили перші три слова перекладеної мети, ми передаємо їх декодеру, який потім використовує всі входи кодера, щоб спробувати передбачити четверте слово.

Щоб пришвидшити роботу під час навчання (коли модель має доступ до цільових речень), декодер подається на всю ціль, але не дозволяється використовувати майбутні слова (якби він мав доступ до слова в позиції 2 під час спроби передбачити слово в позиції 2, проблема була б не дуже складною! Наприклад, під час спроби передбачити четверте слово, шар уваги матиме доступ лише до слів з 1 по 3.

Оригінальна архітектура Трансформеру виглядала так: з кодером ліворуч і декодером праворуч:

Потрібно зауважити, що перший рівень уваги в блоці декодера звертає увагу на всі (попередні) вхідні дані декодера, але другий рівень уваги використовує вихід кодера. Таким чином, він може отримати доступ до всього вхідного речення, щоб найкраще передбачити поточне слово.

Це дуже корисно, оскільки різні мови можуть мати граматичні правила, які встановлюють слова в різному порядку, або деякий контекст, наданий далі в реченні, може допомогти визначити найкращий переклад даного слова.

Маску уваги також можна використовувати в кодері/декодері, щоб не дати моделі звертати увагу на деякі спеціальні слова, наприклад, спеціальне слово доповнення, яке використовується, щоб зробити всі вхідні дані однакової довжини під час групування речень.

[14, 17]

1.17 Токенізатори

Токенізатори є одним із основних компонентів конвеєра ОПМ. Вони служать одній меті: перевести текст у дані, які можна буде обробити моделлю. Моделі можуть обробляти лише числа, тому токенізатори повинні конвертувати введені тексти в числові дані.

У завданнях ОПМ, дані, які зазвичай обробляються, є необробленим текстом. Ось приклад такого тексту: «Джим Хенсон був ляльковим».

Однак моделі можуть обробляти лише числа, тому нам потрібно знайти спосіб перетворити необроблений текст у числа. Це те, що роблять токенізатори, і для цього є багато способів.

Мета полягає в тому, щоб знайти найбільш змістовне уявлення — тобто те, яке має найбільший сенс для моделі.

РОЗДІЛ 2 ІНСТРУМЕНТИ РОЗРОБКИ БОТА ТА НЕЙРОННОЇ МЕРЕЖІ

2.1 Опис предметної області

Для побудови діскорд-бота на основі ШНМ потрібно виділити декілька ключових моментів розробки:

- Стабільний, глибокий алгоритм з яким буде проводитися робота.
- Бібліотека, яка надасть можливість легко використати алгоритм.
- Середа розробка яка може надати не тільки можливість писати код, але й тренувати модель. Іншими словами, надасть можливість використовувати обчислювальні ресурси комп’ютерів гугл.
- API діскорда за допомогою якого бот поширюється в маси.

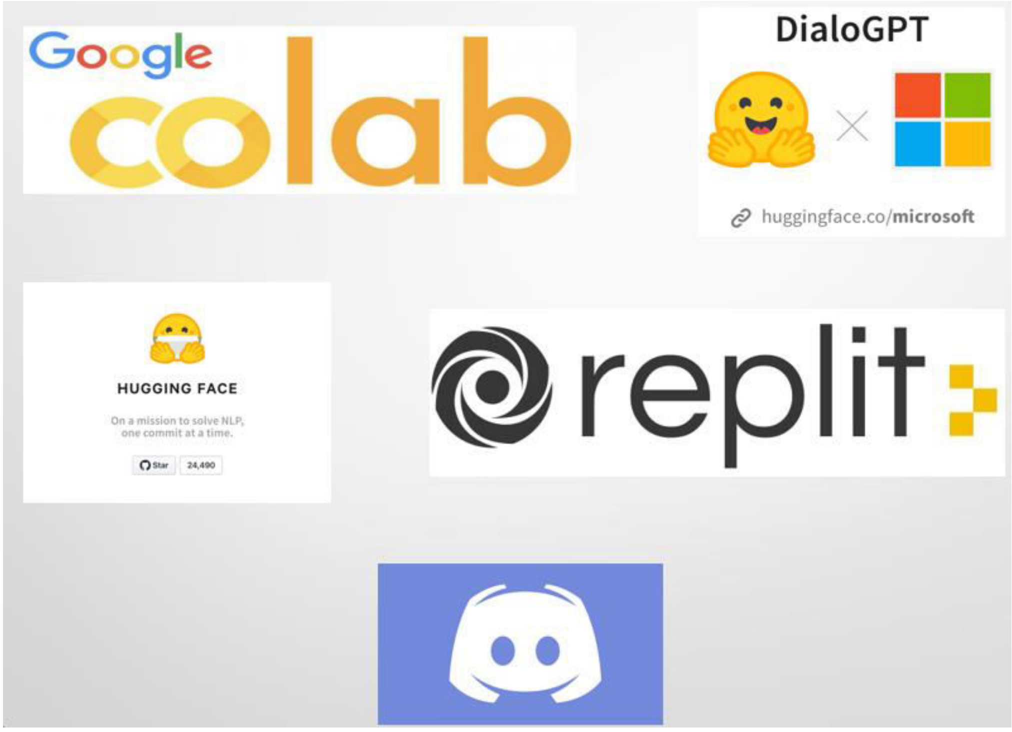


Рис.2.1 Використані технології

Кафедра КІТ

НАУ 22 08 20 − 000 ПЗ

Виконав	*Козловцев Д.С.*			Discord чат-бот на основі алгоритмів обробки природної мови та технологій data science	Літ.	Арк.	Аркушів
Керівник	*Зудов О.М.*					28	6
Консульт.					122 ТП-415Б		
Нормоконт	*Гізун А.І.*						

2.2 Використаний алгоритм ШНМ

Із великого переліку ОПМ моделей було виділено DialoGPT, оскільки цей трансформер не важкий у використанні, важить менше ніж його родичи, має декілька варіацій розміру моделі, але при цьому відпрацьовує на великому рівні ефективності.

Дослідники з Microsoft випустили свою широкомасштабну модель розмови - DialoGPT, яка досягає найсучаснішої продуктивності для генерування відповідної та послідовної відповіді в налаштуваннях діалогової системи. Генерацію відповідей можна розглядати як підпроблему для генерації тексту, де ідея полягає у створенні природного та вільного тексту, який обумовлений та має відношення до послідовності підказок введення.

GPT-2 – це широкомасштабна модель мови на основі трансформатора, яка була навчена на 40 ГБ інтернет-тексту. Це набір декодерних трансформаторів із деякими передовими концепціями навчання, такими як маскована самоувага, кілька головок, залишкові з’єднання, нормалізація шарів тощо, що робить його одним із найкращих генераторів тексту. Мета, яку GPT-2 намагається досягти, полягає в тому, щоб передбачити наступне слово в послідовності, побачивши минулі слова. Реальними застосуваннями для деяких аватарів мовних моделей є функція автозаповнення на смартфонах, автокомпонування в системах електронної пошти тощо.

Автори цієї моделі розповідають про деякі поточні проблеми в системах генерації відповідей і тексту в цілому, до таких проблем відноситься невідповідність стилю, маленька кількість особливостей і комплексності, неможливість охопити довгострокову залежність тощо. Вони вважають, що мовна модель на основі трансформатора, така як GPT-2, яка використовує декілька рівнів замаскованих шарів самоуваги і, як уже було помічено, дуже добре працює з генерацією тексту, потенційно може використовуватись для вирішення деяких з цих обмежень.

Як міру запобіжної попередньої обробки даних, вони (автори моделі) використовують кілька правил обрізання для зразків вводу-виводу перед подачею їх у модель. Нижче наведено 7 кроків попередньої обробки:

- Видалення екземплярів, у яких джерело або цільове значення має URL-адресу.
- Видалення випадків, коли є сліди токсичної мови. Токсичність була виявлена на основі списку заздалегідь визначених слів.
- Видалення випадків, коли відповідь не містила жодного слова зі списку 50 найпоширеніших слів англійської мови. — Цей крок, ймовірно, допоможе підтвердити, що речення написано англійською мовою.
- Видалення екземплярів, у яких відповідь містить спеціальні розмітки, як-от «[« або «]».
- Видалення екземплярів, де довжина джерела і цільового значення сумарно становить більше ніж 200 слів.
- Видалення випадків, коли ціль містить повторення принаймні 3 слів.
- Видалення випадків, коли принаймні 90% триграм зустрічається більше 1000 разів. — допомагає обрізати топорні реакції.

Вони навчали свою модель на 147 мільйонах діалогів створених з ланцюжків коментарів в Reddit з 2005 по 2017 рік. Як стратегію моделювання, автори перетворюють завдання генерації відповідей на завдання вивчення мовної моделі і концептів. Вони розширюють GPT-2 для вирішення цієї проблеми (звідси назва DialoGPT) з припущенням, що цей підхід охопить спільний розподіл P(Target, Source) у потоці розмови з більшою деталізацією.

Вони об’єднують усі діалогові перетворення в межах однієї гілки в довгий текст x1, x2, … , xN, eos, де N — довжина послідовності, а eos — маркер кінця тексту.

Підсумовуючи, DialoGPT адаптує методи попереднього навчання до генерації відповідей, використовуючи сотні гігабайт розмовних даних. Як і GPT-2, DialoGPT спроектована як мовна модель авторегресії (AR) і використовує багаторівневий трансформатор як архітектуру моделі. На відміну від GPT-2, який тренується на загальних текстових даних, DialoGPT використовує 147 мільйонів діалогів, витягнутих з дискусій Reddit.

[18]

2.3 Бібліотека transformers від Hugging Face

Hugging Face це компанія, яка вперше створила додаток чату для нудьгуючих підлітків, вона створює і надає технології ОПМ у відкритий доступ. Два роки назад компанія зібрала 15 мільйонів доларів на створення повної бібліотеки ОПМ. Почавши зі створення чату, Hugging Face швидко розвинула свою експертизу в області обробки мови. Метою компанії є просування концепцій ОПМ та їх демократизації, щоб кожен мав змогу бесплатно і ефективно використовувати найновіші алгоритми цієї області.

Hugging face побудовано на основі концепції моделей трансформаторів, орієнтованих на увагу, і тому не дивно, що ядром екосистеми є їхня бібліотека трансформаторів. Бібліотека трансформаторів підтримується супровідними наборами даних і бібліотеками токенізаторів.

Потрібно пам’ятати, що трансформатори не розуміють текст або будь-які Послідовності тексту у його рідній формі: рядка символів. Швидше, послідовності букв або інші дані повинні бути спочатку перетворені на числову мову векторів, матриць і тензорів. Отже, токенізатор є важливим компонентом будь-якого конвеєра трансформатора.

Hugging Face також надає бібліотеку прискорення, яка легко інтегрується з існуючими навчальними процесами Hugging Face, а також загальними навчальними сценаріями PyTorch, щоб легко розширити можливості розподіленого навчання за допомогою різних пристроїв апаратного прискорення, таких як графічні процесори, TPU тощо. Один і той самий навчальний сценарій можна використовувати під час спеціального навчального запуску з кількома графічними процесорами, спеціалізованим хмарним прискорювачем, таким як TPU, або на ЦП ноутбука для розробки на ходу.

На додаток до трансформаторів, токенізаторів, наборів даних і бібліотек для прискорення, Hugging Face містить ряд ресурсів спільноти. Hugging Face Hub забезпечує організований спосіб обміну власними моделями з іншими і підтримується бібліотекою huggingface_hub.

[16]

2.4 Google Colaboratory

Colaboratory - це безкоштовне середовище для ноутбуків Jupyter, яке повністю працює у хмарному середовищі гугл. Більш важливим є те, що ця середа не вимагає налаштування, а створені в colaboratory блокноти можуть одночасно редагуватися членами команди так само як редагуються документи в Google Docs. Colaboratory підтримує багато популярних бібліотек машинного навчання, які можна легко завантажити на гугл диск або робочу машину.

Colaboratory пропонує:

- Написання та запуск коду на Python
- Документація коду, який підтримує математичні рівняння
- Створення/завантаження/надавання доступу до блокнотів
- Імпорт/збереження блокнотів із/на Google Disc
- Імпорт/публікація блокнотів з GitHub
- Імпорт зовнішніх наборів даних, наприклад з Kaggle

- Інтеграція PyTorch, TensorFlow, Keras, OpenCV
- Безкоштовний хмарний сервіс із безкоштовним графічним процесором

2.2 Діскорд і його API

Діскорд – це платформа для комунікації за допомогою тексту або відео/аудіо дзвінків, яка використовується більшою мірою комп’ютерними гравцями (геймерами). Діскорд має багато додаткових можливостей, в їх число входить стрімінг, відображення музики яку в даний час слухає користувач, відображення активної гри, яку запустив користувач, кастомні ролі та багато чого іншого. Однією з найголовніших можливостей є використання власних або вже готових ботів.

Наразі існує велика кількість різновидів бібліотек на основі API діскорд, сконцентрованих на різні мови програмування. Для даної роботи використовується “discord.py”.

Бот- це автоматизована програма, яка веде себе як користувач, але разом з
тим відповідає на певні команди та івенти, такі програми ще називають бот-юзерами.

Є багато різновидів ботів. Боти виконують елементарні команди по типу виводу певних балів які юзер отримав знаходячись на сервері; вітання нових юзерів; Збір інформації в різних форматах; ітераційне листування з користувачем
за допомогою прописаних фраз; природнє ведення діалогу за допомогою алгоритмів data science і так далі.

Для того щоб створити бота, потрібно:

- Створити бот-юзера в діскорді та зареєструвати його в гільдії
- Написати код боту використовуючи discord API

2.2 Існуючі аналоги

- Akemi-bot: захист сервера, автоматичні ролі, музика
- Infinity Protect: захист серверу
- AkainuBot: економіка, міні-ігри, автоматичні ролі
- Yukikaze: музичний бот, який підтримує Spotify, YouTube, Yandex
- CleverBot: бот підтримує спілкування, а такаож віикористовує ненормативну лексику

Можна побачити, що більшість ботів на данний момент це прості програми, які виконують зазделегіть прописані команди. Є випадки, коли боти побудовані на основі NLP, прикладом слугує CleverBot, який краще розуміє мову і генерує відповіді, ніж HarryBot. Причиною є недостатня кількість даних, які використовувала модель для навчання. Справа в тому, що не дивлячись на комплексність алгоритму, ключовим фактором являється наявність великої кількості профільтрованих даних. Тому логічним продовженням проєкту є здобування данних з серії книг “Гаррі Поттер” і повторного тренування моделі на основі нової інформації.

РОЗДІЛ 3 ПОБУДОВА АЛГОРИТМУ ТА ДІСКОРД-БОТА

3.1 Початок роботи з Google Colaboratory

Перш за все, потрібно завантажити файл з розширенням model_train_upload_workflow.ipynb, в якому буде написаний код, на гугл диск. Після цього потрібно відкрити цей файл, натиснувши на нього правою кнопкою миші та вибравши у випадаючому меню Colaboratory.

Colaboratory надає можливість не тільки зручно писати код, але й використовувати гугл GPU, що дуже важливо для тренування моделі.

Для того щоб використовувати завантажені файли в середовищу Colaboratory, потрібно підключитися до гугл-диску

```
[ ] from google.colab import drive
    drive.mount('/content/drive/')

    Mounted at /content/drive/

[ ] !pip -q install transformers

                   |████████████████████████████████| 4.2 MB 5.2 MB/s
                   |████████████████████████████████| 6.6 MB 51.4 MB/s
                   |████████████████████████████████| 596 kB 56.2 MB/s
                   |████████████████████████████████| 86 kB 6.4 MB/s

[ ] import os
    os.chdir("/content/drive/My Drive")
```

Рис.3.1. Підключення до гугл-диску в Colaboratory

Рядок !pip –q install transformers завантажує бібліотеку з моделями, всі наступні використані пакеджи наведено у додатку з кодом.

Після підключення до гугл-диску, потрібно підключитись Kaggle, сайту для розробників алгоритмів машинного навчання, де зберігається велика кількість наборів даних, та зчитати вибраний нами набір за допомогою бібліотеки pandas.

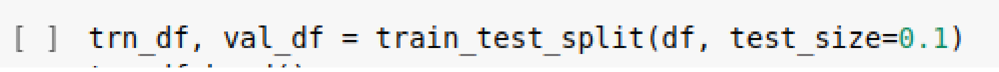
```
[ ] !mkdir ~/.kaggle
    !cp kaggle.json ~/.kaggle/kaggle.json

[ ] !kaggle datasets download gulsahdemiryurek/harry-potter-dataset -f "Harry Potter 1.csv"

    Harry%20Potter%201.csv: Skipping, found more recently modified local copy (use --force to force download)

[ ] data = pd.read_csv('Harry%20Potter%201.csv', sep=';')
```

Рис.3.2. Підключення до Kaggle та зчитування даних

“Harry Potter 1.csv” – це чистий набір даних, завдяки цьому ми зразу можемо перейти до роботи, без проведення попередньої фільтрації.

3.2 Підготовка даних до роботи з моделлю

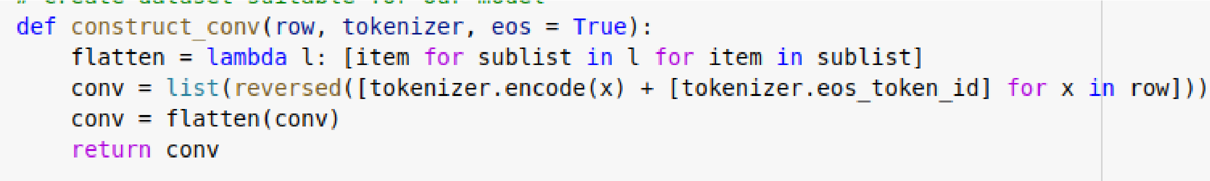
Загальною практикою є поділ даних на дві частини, які включають в себе дані для тренування та валідації у відношені 9 до 1.



```
[ ]  trn_df, val_df = train_test_split(df, test_size=0.1)
```

Рис.3.3. Ділення даних

Оскільки всі моделі приймають числові значення, потрібно перевести дані в відповідний формат, токенізувавши кожне слово, таким шляхом, щоб речення вміщалось у математичний вектор, який буде подаватися на вхід алгоритму.



```
def construct_conv(row, tokenizer, eos = True):
    flatten = lambda l: [item for sublist in l for item in sublist]
    conv = list(reversed([tokenizer.encode(x) + [tokenizer.eos_token_id] for x in row]))
    conv = flatten(conv)
    return conv
```

Рис.3.4. Токенізація даних

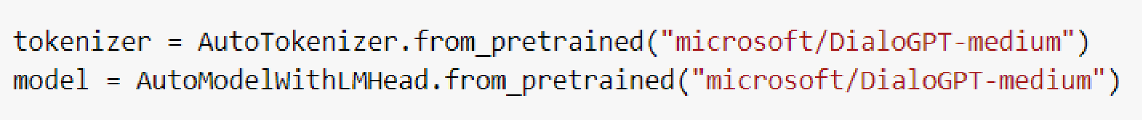
Оскільки моделі типу трансформерів доволі великі- потрібні так звані чекпоінти, що дозволяють зберігати стан моделі на різних етапах навчання. Це дозволяє при помилці в навчанні почати навчання перед помилкою, тим самим зберігаючи час і ресурси. Зберігатися чекпоінти, як і дані будуть в файлі створеному за допомогою технології pickle, яка дозволяє швидко отримати інформацію коли потрібно.

```
 class ConversationDataset(Dataset):

def load_and_cache_examples(args, tokenizer, df_trn, df_val, evaluate=False):
   return ConversationDataset(tokenizer, args, df_val if evaluate else df_trn)

def set_seed(args):
def  sorted checkpoints(args, checkpoint prefix="checkpoint", use mtime=False) -> List[str]:
def _rotate_checkpoints(args, checkpoint_prefix="checkpoint", use_mtime=False) -> None:
```

Рис.3.5. Клас і функції для генерації токенізованого набору даних з чекпоінтами

3.3 Побудова, тренування і оцінка моделі

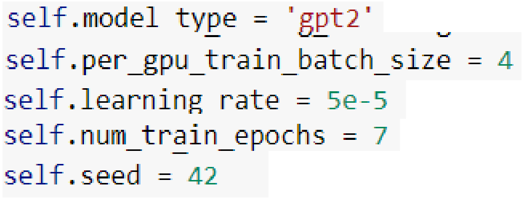
Після імпорту потрібних класів з бібліотеки трансформатерів, потрібно ініціалізувати модель та прописати все важливі параметри.



```
tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-medium")
model = AutoModelWithLMHead.from_pretrained("microsoft/DialoGPT-medium")
```

Рис.3.6 Ініціалізація моделі

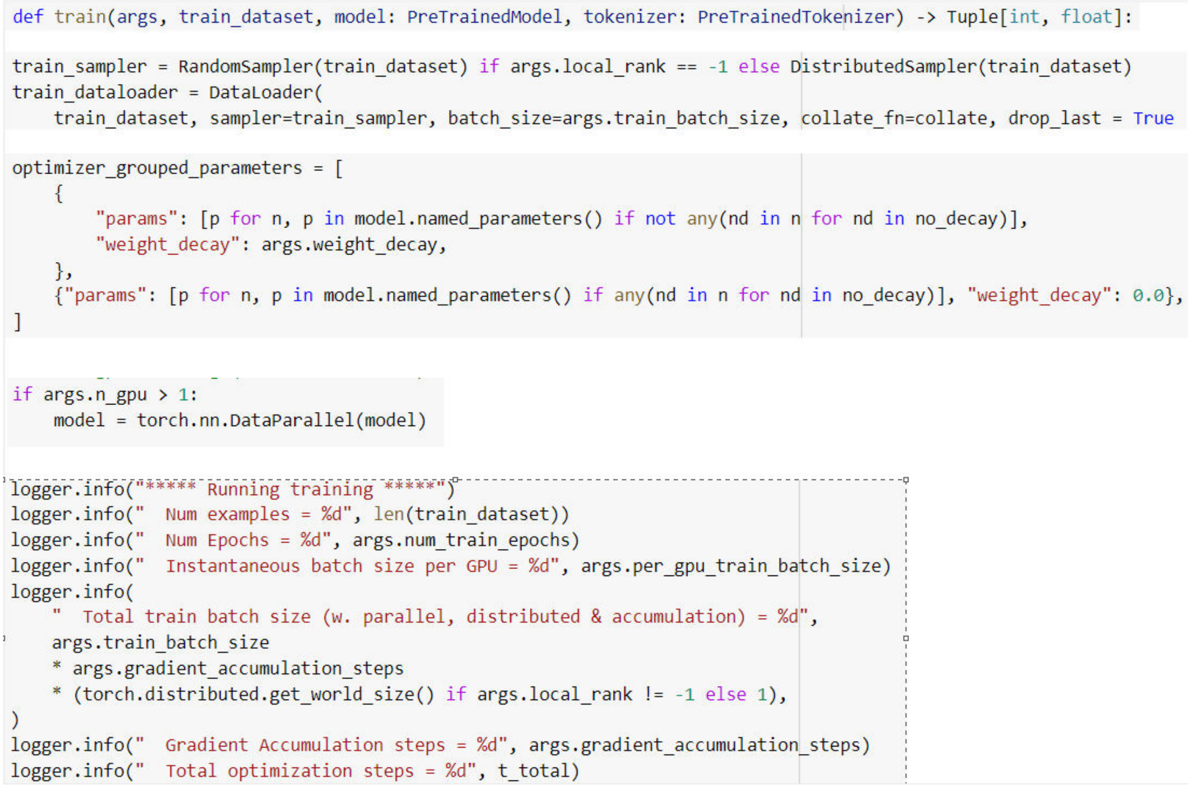
З найважливіших параметрів можна виділити параметр, який задає типаж моделі, оскільки трансформаторів є декілька; розмір батчу, який задає кількість даних що тренуються в одну епоху; навчальний темп який задає швидкість навчання моделі за епоху; кількість епох, тобто разів проходу всього алгоритму по даним з вираховуванням помилок та їх подальшої корекції; параметр, який зберігає вивід генератора випадковості і робить його спільним для всіх функцій, де був прописаний.



```
self.model type = 'gpt2'
self.per_gpu_train_batch_size = 4
self.learning rate = 5e-5
self.num_train_epochs = 7
self.seed = 42
```

Рис.3.7 Цікаві параметри

Для тренування моделі відведена особлива функція train яка включає в себе функції і стейтменти, відповідаючі за семплінг по батчам, оптимізацію даних, контроль GPU та саме тренування.



```
def train(args, train_dataset, model: PreTrainedModel, tokenizer: PreTrainedTokenizer) -> Tuple[int, float]:

train_sampler = RandomSampler(train_dataset) if args.local_rank == -1 else DistributedSampler(train_dataset)
train_dataloader = DataLoader(
    train_dataset, sampler=train_sampler, batch_size=args.train_batch_size, collate_fn=collate, drop_last = True

optimizer_grouped_parameters = [
    {
        "params": [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)],
        "weight_decay": args.weight_decay,
    },
    {"params": [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)], "weight_decay": 0.0},
]


if args.n_gpu > 1:
    model = torch.nn.DataParallel(model)


logger.info("***** Running training *****")
logger.info("  Num examples = %d", len(train_dataset))
logger.info("  Num Epochs = %d", args.num_train_epochs)
logger.info("  Instantaneous batch size per GPU = %d", args.per_gpu_train_batch_size)
logger.info(
    "  Total train batch size (w. parallel, distributed & accumulation) = %d",
    args.train_batch_size
    * args.gradient_accumulation_steps
    * (torch.distributed.get_world_size() if args.local_rank != -1 else 1),
)
logger.info("  Gradient Accumulation steps = %d", args.gradient_accumulation_steps)
logger.info("  Total optimization steps = %d", t_total)
```

Рис.3.8 Головні строки відповідаючі за семплінг, оптимізацію даних, контроль GPU та тренування моделі

Останнім кроком є оцінка моделі, яка здійснюється у функції evaluate() та визначається змінною perplexity. Чим нижче даний коефіцієнт, тим краще справилась модель. Коли вся робота закінчена, прописується функція main, де ми викликаємо всі раніше прописані функції.

```
>> User:Hello Harry
HarryBot: It's me.
>> User:You are a wizard
HarryBot: I'm Harry.
>> User:Yes, yes you are
HarryBot: I am.
>> User:Let's go to Hogwarts
HarryBot: We'll take him.
```

```
>> User:Hi
HarryBot: Hello, Hagrid.
>> User:Harry, you are chosen one
HarryBot: I'm not Harry.
>> User:Who are you then?
HarryBot: I don't know.
>> User:You are a wizard
HarryBot: What's your name?
```

```
>> User:Hi Harry, it's good to see you.
HarryBot: Oh, nice to meet you.
>> User:How was your day?
HarryBot: You all right, Harry?
>> User:I'm not Harry, my name is Dima
HarryBot: You're a bit behind schedule.
>> User:Right we have a quidditch match soon.
HarryBot: I think we'll need another one.
```

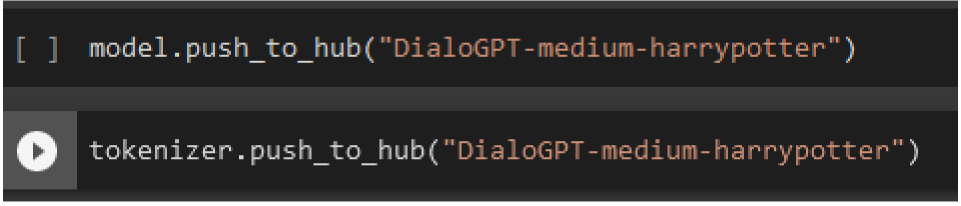
```
>> User:Hello young wizzard
HarryBot: Hello, Harry.
>> User:I'm not Harry, you are
HarryBot: Sorry about that.
>> User:That's ok, want to go to school?
HarryBot: No, thanks.
>> User:I thought you liked school
HarryBot: Sorry, Harry, didn't mean to make you go there.
```

```
>> User:Hello Harry, how are you?
HarryBot: You're a wizard, Harry.
>> User:That's that Hagrid has said to you.
HarryBot: Yes, I am a what?
>> User:A wizard.
HarryBot: A what?
>> User:Wizard.
HarryBot: !!!?!!What!!
```

Рис.3.9 Приклади роботи алгоритму

3.4 Завантаження моделі до Huggingface

Huggingface надає можливість зберегти все, що вивчила модель і запускати алгоритм коли завгодно, без постійного тренування. Також Huggingface API може використовуватись для того, щоб підключити модель до боту. Для завантаження моделі, потрібно через консоль залогінитися в акаунт на даній платформі, створити репозиторій та додати файли. Закінчивши всі процедури, можна завантажувати модель і токенайзер.



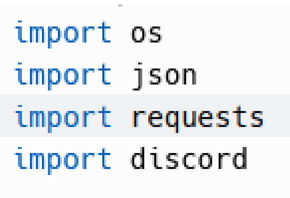
```
[ ]  model.push_to_hub("DialoGPT-medium-harrypotter")
```

```
tokenizer.push_to_hub("DialoGPT-medium-harrypotter")
```

Рис.3.10 Завантаження алгоритму і токенізатору до Huggingace

3.5 Створення боту на основі API діскорду

З самого початку потрібно завантажити потрібні модулі: os надає можливість користуватися змінними середи, такими як токени діскорду та Haggingface; json, request для запитів (повідомлень) які будуть надсилатись боту; і, звичайно, discord, за допомогою якого ми можемо користуватись API діскорду.

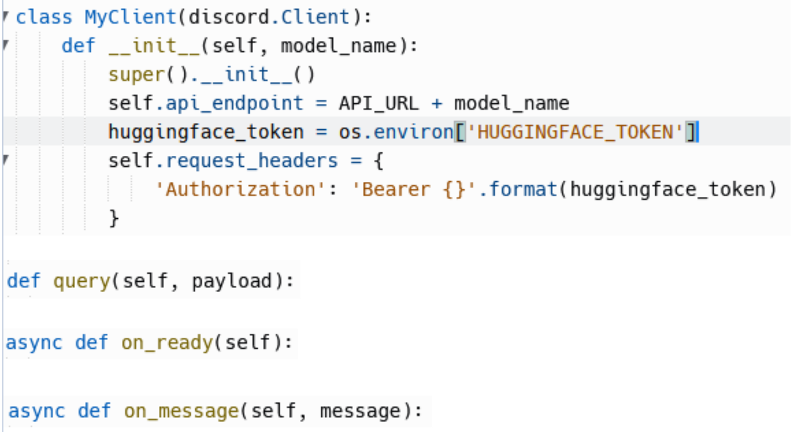


```
import os
import json
import requests
import discord
```

Рис.3.11 Завантаження модулей

Для з’єднання з Huggingface потрібно надати в глобальну змінну API_URL посилання на /models/username/ в самому середовищі. Для того, щоб скрипт брав потрібні файли, пізніше додаються ще декілька уточнюючих деталей.

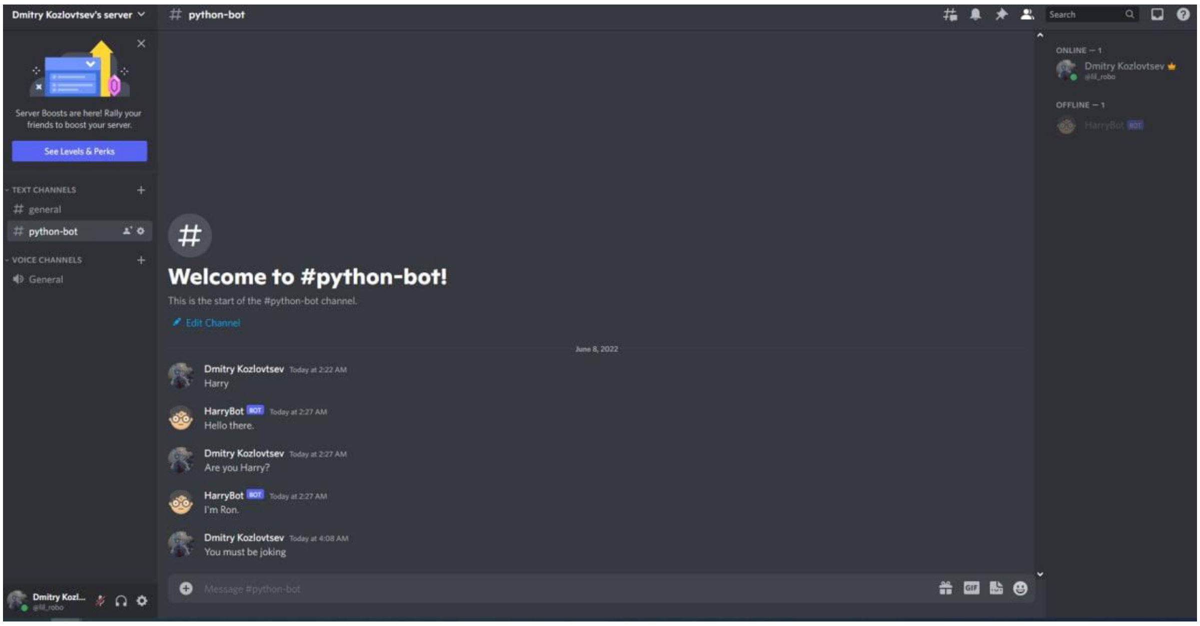
Весь функціонал прописується в класі MyClient, де уточнюється яку саме модель з профілю девелопера на Huggingface потрібно використати. Функція queue відповідає за відпрацювання запитів які надходять до бота. Функція on_ready показує стан боту: онлайн чи офлайн. Функція on_message відслідковує повідомлення на каналі та генерує відповідь.



```
class MyClient(discord.Client):
    def __init__(self, model_name):
        super().__init__()
        self.api_endpoint = API_URL + model_name
        huggingface_token = os.environ['HUGGINGFACE_TOKEN']
        self.request_headers = {
            'Authorization': 'Bearer {}'.format(huggingface_token)
        }

def query(self, payload):

async def on_ready(self):

async def on_message(self, message):
```

Рис.3.12 Функціонал боту прописаний в класі MyClient

Під кінець, в функції main, викликається описаний вище клас, а бот починає працювати в діскорд. Відтак, отримуємо результат:



June 8, 2022

Dmitry Kozlovtsev Today at 2:22 AM
Harry

HarryBot BOT Today at 2:27 AM
Hello there.

Dmitry Kozlovtsev Today at 2:27 AM
Are you Harry?

HarryBot BOT Today at 2:27 AM
I'm Ron.

Dmitry Kozlovtsev Today at 4:08 AM
You must be joking

Рис.3.13 Результат роботи

[19-29]

ВИСНОВОК

Дипломний проект на тему «Discord чат-бот на основі алгоритмів обробки природної мови та технологій data science» викликав великий інтерес у широго кола моїх знайомих, які користуються послугами діскорду та проводять велику кількість часу в відео-чатах.

Особливість проєкту заключається в його доступності всім зацікавленим в ШНМ та їх застосуваннях. Юзеру не потрібно писати код та знаходити середу, де можна навчити модель та поспілкуватись з нею, достатньо бути зареєстрованим в діскорд. Але, на відміну від більшості ботів в діскорд, HarryBot надає юзеру веселе і, найголовніше, живе спілкування, а не завчені відповіді або відповіді на прописані команди.

За допомогою сучасних методів розробки моделей, таких як трансформер DialoGPT був створений HarryBot, який, хоч і не підтримує логічної розмови весь час, але повністю виконує свою функцію як програма для розваги та популізації ШНМ.

API діскорду та проектування програми забезпечує безперебійне функціонування бота, швидкий і точний обмін повідомленнями з великою кількістю юзерів, до тисячи одночасно, якщо бути точним.

В тестуванні приймало участь зацікавлене коло геймерів, друзів, знайомих та звичайних юзерів діскорду. Згідно отриманих реценнзій можна зробити висновок, що бот працює швидко і без перебоїв, але не надто розумний.

Планується покращення бота шляхом додавання реплік персонажів із всіх книг популярної серії. Велика кількість даних виведе роботу бота на новий рівень, де, можливо, буде важко відрізнити його від реальної людини.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. https://futurum.today/shtuchni-neironni-merezhi-shcho-tse-take/
2. https://www.poznavayka.org/uk/nauka-i-tehnika-2/neyronni-merezhi-yih-zastosuvannya-robota/
3. https://tmginfo.net/2020/06/shtuchni-neyronni-merezhi-yak-voni-zmozhut/
4. Андрій Бурков. The Hundred-Page Machine Learning Book: Андрій Бурков, 2019. – 160 с.
5. Aurélien Géron. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques: "O’REILLY", 2019. - 801 с.
6. Marc Peter Deisenroth, A. Aldo Faisal, and Cheng Soon Ong. Mathematics for Machine Learning: Published by Cambridge University Press, 2020. – 367 с.
7. Oliver Theobald. Machine Learning For Absolute Beginners: A Plain English Introduction (Second Edition) (Machine Learning From Scratch Book 1): Oliver Theobald, 2017 - 164 с.
8. Andrew Trask. Grokking Deep Learning: Manning Publications Co, 2019-301с.
9. О.М. Різник, Динамічні рекурентні нейронні мережі: Інститут проблем математичних машин та систем Національної академії наук України, 2009. – 26 с.
10. https://www.geeksforgeeks.org/introduction-to-recurrent-neural-network/
11. https://towardsdatascience.com/illustrated-guide-to-recurrent-neural-networks-79e5eb8049c9
12. В. О. Лебедєв, Модель згорткової нейронної мережі для обробки великих зображень: м-во освіти і науки України, 2019. – 90 с.
13. https://evergreens.com.ua/ua/articles/cnn.html
14. https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04
15. https://towardsdatascience.com/illustrated-guide-to-transformers-step-by-step-explanation-f74876522bc0
16. https://huggingface.co/docs/transformers/index

17. https://jalammar.github.io/illustrated-transformer/
18. https://huggingface.co/docs/transformers/model_doc/dialogpt
19. https://www.ringcentral.co.uk/gb/en/blog/the-definitive-guide-to-using-a-deep-learning-or-machine-learning-chatbot-for-your-business/
20. https://shanebarker.com/blog/deep-learning-chatbot/
21. https://towardsdatascience.com/creating-a-smart-chat-bot-that-talks-like-you-79bb700b288f
22. https://towardsdatascience.com/how-to-build-your-own-chatbot-using-deep-learning-bb41f970e281
23. https://www.analyticsvidhya.com/blog/2021/10/complete-guide-to-build-your-ai-chatbot-with-nlp-in-python/
24. https://www.altexsoft.com/blog/datascience/a-technological-guide-to-building-an-ai-chatbot/
25. https://towardsdatascience.com/how-to-create-a-chatbot-with-python-deep-learning-in-less-than-an-hour-56a063bdfc44
26. https://www.cleveroad.com/blog/how-to-make-a-chatbot
27. https://www.chatbot.com/help/build-your-chatbot/how-to-build-your-chatbot/
28. https://www.mygreatlearning.com/blog/basics-of-building-an-artificial-intelligence-chatbot/
29. https://www.udemy.com/course/chatbot/
30. Положення про дипломні роботи (проекти) випускників Національного Авіаційного Університету Київ 2017. -72

ДОДАТКИ

```
# %% [markdown]
# # Fine-tune a DialoGPT model

# %% [markdown]
# ## Setup

# %%
from google.colab import drive
drive.mount('/content/drive/')

# %%
!pip -q install transformers

# %%
import os
os.chdir("/content/drive/My Drive")

# %%
# all the imports

import glob
import logging
import os
import pickle
import random
import re
import shutil
from typing import Dict, List, Tuple

import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split

from torch.nn.utils.rnn import pad_sequence
from torch.utils.data import DataLoader, Dataset, RandomSampler, SequentialSampler
from torch.utils.data.distributed import DistributedSampler
from tqdm.notebook import tqdm, trange

from pathlib import Path

from transformers import (
    MODEL_WITH_LM_HEAD_MAPPING,
    WEIGHTS_NAME,
    AdamW,
    AutoConfig,
    PreTrainedModel,
    PreTrainedTokenizer,
    get_linear_schedule_with_warmup,
)
```

```
try:
    from torch.utils.tensorboard import SummaryWriter
except ImportError:
    from tensorboardX import SummaryWriter

# %% [markdown]
# ## Get Data from Kaggle

# %%
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle/kaggle.json

# %%
!kaggle datasets download gulsahdemiryurek/harry-potter-dataset -f "Harry Potter 1.csv"

# %%
data = pd.read_csv('Harry%20Potter%201.csv', sep=';')

# %%
data.sample(6)

# %%
data.rename(columns={'Character': 'name', 'Sentence': 'line'}, inplace=True)

# %%
len(data)

# %%
sum(data.name == 'Harry')

# %%
CHARACTER_NAME = 'Harry'

# %%
contexted = []

n = 7

for i in data[data.name == CHARACTER_NAME].index:
  if i < n:
    continue
  row = []
  prev = i - 1 - n
  for j in range(i, prev, -1):
    row.append(data.line[j])
  contexted.append(row)

columns = ['response', 'context']
columns = columns + ['context/' + str(i) for i in range(n - 1)]

df = pd.DataFrame.from_records(contexted, columns=columns)
```

```
# %%
df.sample(6)

# %%
trn_df, val_df = train_test_split(df, test_size=0.1)
trn_df.head()

# %%
# create dataset suitable for our model
def construct_conv(row, tokenizer, eos = True):
    flatten = lambda l: [item for sublist in l for item in sublist]
    conv = list(reversed([tokenizer.encode(x) + [tokenizer.eos_token_id] for x in row]))
    conv = flatten(conv)
    return conv

class ConversationDataset(Dataset):
    def __init__(self, tokenizer: PreTrainedTokenizer, args, df, block_size=512):

        block_size = block_size - (tokenizer.model_max_length - tokenizer.max_len_single_sentence)

        directory = args.cache_dir
        cached_features_file = os.path.join(
            directory, args.model_type + "_cached_lm_" + str(block_size)
        )

        if os.path.exists(cached_features_file) and not args.overwrite_cache:
            logger.info("Loading features from cached file %s", cached_features_file)
            with open(cached_features_file, "rb") as handle:
                self.examples = pickle.load(handle)
        else:
            logger.info("Creating features from dataset file at %s", directory)

            self.examples = []
            for _, row in df.iterrows():
                conv = construct_conv(row, tokenizer)
                self.examples.append(conv)

            logger.info("Saving features into cached file %s", cached_features_file)
            with open(cached_features_file, "wb") as handle:
                pickle.dump(self.examples, handle, protocol=pickle.HIGHEST_PROTOCOL)

    def __len__(self):
        return len(self.examples)

    def __getitem__(self, item):
        return torch.tensor(self.examples[item], dtype=torch.long)

# %%
# Cacheing and storing of data/checkpoints

def load_and_cache_examples(args, tokenizer, df_trn, df_val, evaluate=False):
    return ConversationDataset(tokenizer, args, df_val if evaluate else df_trn)
```

```
def set_seed(args):
    random.seed(args.seed)
    np.random.seed(args.seed)
    torch.manual_seed(args.seed)
    if args.n_gpu > 0:
        torch.cuda.manual_seed_all(args.seed)


def _sorted_checkpoints(args, checkpoint_prefix="checkpoint", use_mtime=False) -> List[str]:
    ordering_and_checkpoint_path = []

    glob_checkpoints = glob.glob(os.path.join(args.output_dir, "{}-*".format(checkpoint_prefix)))

    for path in glob_checkpoints:
        if use_mtime:
            ordering_and_checkpoint_path.append((os.path.getmtime(path), path))
        else:
            regex_match = re.match(".*{}-([0-9]+)".format(checkpoint_prefix), path)
            if regex_match and regex_match.groups():
                ordering_and_checkpoint_path.append((int(regex_match.groups()[0]), path))

    checkpoints_sorted = sorted(ordering_and_checkpoint_path)
    checkpoints_sorted = [checkpoint[1] for checkpoint in checkpoints_sorted]
    return checkpoints_sorted


def _rotate_checkpoints(args, checkpoint_prefix="checkpoint", use_mtime=False) -> None:
    if not args.save_total_limit:
        return
    if args.save_total_limit <= 0:
        return

    # Check if we should delete older checkpoint(s)
    checkpoints_sorted = _sorted_checkpoints(args, checkpoint_prefix, use_mtime)
    if len(checkpoints_sorted) <= args.save_total_limit:
        return

    number_of_checkpoints_to_delete = max(0, len(checkpoints_sorted) - args.save_total_limit)
    checkpoints_to_be_deleted = checkpoints_sorted[:number_of_checkpoints_to_delete]
    for checkpoint in checkpoints_to_be_deleted:
        logger.info("Deleting older checkpoint [{}] due to args.save_total_limit".format(checkpoint))
        shutil.rmtree(checkpoint)

# %% [markdown]
# ## Build Model

# %%
from transformers import AutoModelWithLMHead, AutoModelForCausalLM, AutoTokenizer
import torch

tokenizer = AutoTokenizer.from_pretrained("microsoft/DialoGPT-medium")
model = AutoModelWithLMHead.from_pretrained("microsoft/DialoGPT-medium")
```

```
# %%
"""
Fine-tuning the library models for language modeling on a text file (GPT, GPT-2, BERT,
RoBERTa).
GPT and GPT-2 are fine-tuned using a causal language modeling (CLM) loss while BERT and
RoBERTa are fine-tuned
using a masked language modeling (MLM) loss.
"""

# Configs
logger = logging.getLogger(__name__)

MODEL_CONFIG_CLASSES = list(MODEL_WITH_LM_HEAD_MAPPING.keys())
MODEL_TYPES = tuple(conf.model_type for conf in MODEL_CONFIG_CLASSES)

# %%
# Args to allow for easy convertion of python script to notebook
class Args():
    def __init__(self):
        self.output_dir = 'output-small'
        self.model_type = 'gpt2'
        self.model_name_or_path = 'microsoft/DialoGPT-small'
        self.config_name = 'microsoft/DialoGPT-small'
        self.tokenizer_name = 'microsoft/DialoGPT-small'
        self.cache_dir = 'cached'
        self.block_size = 512
        self.do_train = True
        self.do_eval = True
        self.evaluate_during_training = False
        self.per_gpu_train_batch_size = 4
        self.per_gpu_eval_batch_size = 4
        self.gradient_accumulation_steps = 1
        self.learning_rate = 5e-5
        self.weight_decay = 0.0
        self.adam_epsilon = 1e-8
        self.max_grad_norm = 1.0
        self.num_train_epochs = 7
        self.max_steps = -1
        self.warmup_steps = 0
        self.logging_steps = 1000
        self.save_steps = 3500
        self.save_total_limit = None
        self.eval_all_checkpoints = False
        self.no_cuda = False
        self.overwrite_output_dir = True
        self.overwrite_cache = True
        self.should_continue = False
        self.seed = 42
        self.local_rank = -1
        self.fp16 = False
        self.fp16_opt_level = 'O1'
```

```
args = Args()

# %% [markdown]
# ## Train and Evaluate

# %%
def train(args, train_dataset, model: PreTrainedModel, tokenizer: PreTrainedTokenizer) ->
Tuple[int, float]:
    """ Train the model """
    if args.local_rank in [-1, 0]:
        tb_writer = SummaryWriter()

    args.train_batch_size = args.per_gpu_train_batch_size * max(1, args.n_gpu)

    def collate(examples: List[torch.Tensor]):
        if tokenizer._pad_token is None:
            return pad_sequence(examples, batch_first=True)
        return pad_sequence(examples, batch_first=True, padding_value=tokenizer.pad_token_id)

    train_sampler = RandomSampler(train_dataset) if args.local_rank == -1 else
DistributedSampler(train_dataset)
    train_dataloader = DataLoader(
        train_dataset, sampler=train_sampler, batch_size=args.train_batch_size, collate_fn=collate,
drop_last = True
    )

    if args.max_steps > 0:
        t_total = args.max_steps
        args.num_train_epochs = args.max_steps // (len(train_dataloader) //
args.gradient_accumulation_steps) + 1
    else:
        t_total = len(train_dataloader) // args.gradient_accumulation_steps * args.num_train_epochs

    model = model.module if hasattr(model, "module") else model  # Take care of
distributed/parallel training
    model.resize_token_embeddings(len(tokenizer))
    # add_special_tokens_(model, tokenizer)


    # Prepare optimizer and schedule (linear warmup and decay)
    no_decay = ["bias", "LayerNorm.weight"]
    optimizer_grouped_parameters = [
        {
            "params": [p for n, p in model.named_parameters() if not any(nd in n for nd in no_decay)],
            "weight_decay": args.weight_decay,
        },
        {"params": [p for n, p in model.named_parameters() if any(nd in n for nd in no_decay)],
"weight_decay": 0.0},
    ]
    optimizer = AdamW(optimizer_grouped_parameters, lr=args.learning_rate,
eps=args.adam_epsilon)
    scheduler = get_linear_schedule_with_warmup(
        optimizer, num_warmup_steps=args.warmup_steps, num_training_steps=t_total
```

```
    )

    # Check if saved optimizer or scheduler states exist
    if (
        args.model_name_or_path
        and os.path.isfile(os.path.join(args.model_name_or_path, "optimizer.pt"))
        and os.path.isfile(os.path.join(args.model_name_or_path, "scheduler.pt"))
    ):
        # Load in optimizer and scheduler states
        optimizer.load_state_dict(torch.load(os.path.join(args.model_name_or_path, "optimizer.pt")))
        scheduler.load_state_dict(torch.load(os.path.join(args.model_name_or_path, "scheduler.pt")))

    if args.fp16:
        try:
            from apex import amp
        except ImportError:
            raise ImportError("Please install apex from https://www.github.com/nvidia/apex to use fp16
training.")
        model, optimizer = amp.initialize(model, optimizer, opt_level=args.fp16_opt_level)

    # multi-gpu training (should be after apex fp16 initialization)
    if args.n_gpu > 1:
        model = torch.nn.DataParallel(model)

    # Distributed training (should be after apex fp16 initialization)
    if args.local_rank != -1:
        model = torch.nn.parallel.DistributedDataParallel(
            model, device_ids=[args.local_rank], output_device=args.local_rank,
find_unused_parameters=True
        )

    # Train!
    logger.info("***** Running training *****")
    logger.info("  Num examples = %d", len(train_dataset))
    logger.info("  Num Epochs = %d", args.num_train_epochs)
    logger.info("  Instantaneous batch size per GPU = %d", args.per_gpu_train_batch_size)
    logger.info(
        "  Total train batch size (w. parallel, distributed & accumulation) = %d",
        args.train_batch_size
        * args.gradient_accumulation_steps
        * (torch.distributed.get_world_size() if args.local_rank != -1 else 1),
    )
    logger.info("  Gradient Accumulation steps = %d", args.gradient_accumulation_steps)
    logger.info("  Total optimization steps = %d", t_total)

    global_step = 0
    epochs_trained = 0
    steps_trained_in_current_epoch = 0
    # Check if continuing training from a checkpoint
    if args.model_name_or_path and os.path.exists(args.model_name_or_path):
        try:
            # set global_step to gobal_step of last saved checkpoint from model path
            checkpoint_suffix = args.model_name_or_path.split("-")[-1].split("/")[0]
```

```
            global_step = int(checkpoint_suffix)
            epochs_trained = global_step // (len(train_dataloader) // args.gradient_accumulation_steps)
            steps_trained_in_current_epoch = global_step % (len(train_dataloader) //
args.gradient_accumulation_steps)

            logger.info("  Continuing training from checkpoint, will skip to saved global_step")
            logger.info("  Continuing training from epoch %d", epochs_trained)
            logger.info("  Continuing training from global step %d", global_step)
            logger.info("  Will skip the first %d steps in the first epoch",
steps_trained_in_current_epoch)
        except ValueError:
            logger.info("  Starting fine-tuning.")

    tr_loss, logging_loss = 0.0, 0.0

    model.zero_grad()
    train_iterator = trange(
        epochs_trained, int(args.num_train_epochs), desc="Epoch", disable=args.local_rank not in [-1,
0]
    )
    set_seed(args)  # Added here for reproducibility
    for _ in train_iterator:
        epoch_iterator = tqdm(train_dataloader, desc="Iteration", disable=args.local_rank not in [-1,
0])
        for step, batch in enumerate(epoch_iterator):

            # Skip past any already trained steps if resuming training
            if steps_trained_in_current_epoch > 0:
                steps_trained_in_current_epoch -= 1
                continue

            inputs, labels = (batch, batch)
            if inputs.shape[1] > 1024: continue
            inputs = inputs.to(args.device)
            labels = labels.to(args.device)
            model.train()
            outputs = model(inputs, labels=labels)
            loss = outputs[0]  # model outputs are always tuple in transformers (see doc)

            if args.n_gpu > 1:
                loss = loss.mean()  # mean() to average on multi-gpu parallel training
            if args.gradient_accumulation_steps > 1:
                loss = loss / args.gradient_accumulation_steps

            if args.fp16:
                with amp.scale_loss(loss, optimizer) as scaled_loss:
                    scaled_loss.backward()
            else:
                loss.backward()

            tr_loss += loss.item()
            if (step + 1) % args.gradient_accumulation_steps == 0:
                if args.fp16:
```

```
                    torch.nn.utils.clip_grad_norm_(amp.master_params(optimizer), args.max_grad_norm)
                else:
                    torch.nn.utils.clip_grad_norm_(model.parameters(), args.max_grad_norm)
                optimizer.step()
                scheduler.step()  # Update learning rate schedule
                model.zero_grad()
                global_step += 1

                if args.local_rank in [-1, 0] and args.logging_steps > 0 and global_step %
args.logging_steps == 0:
                    # Log metrics
                    if (
                        args.local_rank == -1 and args.evaluate_during_training
                    ):  # Only evaluate when single GPU otherwise metrics may not average well
                        results = evaluate(args, model, tokenizer)
                        for key, value in results.items():
                            tb_writer.add_scalar("eval_{}".format(key), value, global_step)
                    tb_writer.add_scalar("lr", scheduler.get_lr()[0], global_step)
                    tb_writer.add_scalar("loss", (tr_loss - logging_loss) / args.logging_steps, global_step)
                    logging_loss = tr_loss

                if args.local_rank in [-1, 0] and args.save_steps > 0 and global_step % args.save_steps ==
0:
                    checkpoint_prefix = "checkpoint"
                    # Save model checkpoint
                    output_dir = os.path.join(args.output_dir, "{}-{}".format(checkpoint_prefix,
global_step))
                    os.makedirs(output_dir, exist_ok=True)
                    model_to_save = (
                        model.module if hasattr(model, "module") else model
                    )  # Take care of distributed/parallel training
                    model_to_save.save_pretrained(output_dir)
                    tokenizer.save_pretrained(output_dir)

                    torch.save(args, os.path.join(output_dir, "training_args.bin"))
                    logger.info("Saving model checkpoint to %s", output_dir)

                    _rotate_checkpoints(args, checkpoint_prefix)

                    torch.save(optimizer.state_dict(), os.path.join(output_dir, "optimizer.pt"))
                    torch.save(scheduler.state_dict(), os.path.join(output_dir, "scheduler.pt"))
                    logger.info("Saving optimizer and scheduler states to %s", output_dir)

            if args.max_steps > 0 and global_step > args.max_steps:
                epoch_iterator.close()
                break
        if args.max_steps > 0 and global_step > args.max_steps:
            train_iterator.close()
            break

    if args.local_rank in [-1, 0]:
        tb_writer.close()
```

```
    return global_step, tr_loss / global_step

# Evaluation of some model

def evaluate(args, model: PreTrainedModel, tokenizer: PreTrainedTokenizer, df_trn, df_val,
prefix="") -> Dict:
    # Loop to handle MNLI double evaluation (matched, mis-matched)
    eval_output_dir = args.output_dir

    eval_dataset = load_and_cache_examples(args, tokenizer, df_trn, df_val, evaluate=True)
    os.makedirs(eval_output_dir, exist_ok=True)
    args.eval_batch_size = args.per_gpu_eval_batch_size * max(1, args.n_gpu)
    # Note that DistributedSampler samples randomly

    def collate(examples: List[torch.Tensor]):
        if tokenizer._pad_token is None:
            return pad_sequence(examples, batch_first=True)
        return pad_sequence(examples, batch_first=True, padding_value=tokenizer.pad_token_id)

    eval_sampler = SequentialSampler(eval_dataset)
    eval_dataloader = DataLoader(
        eval_dataset, sampler=eval_sampler, batch_size=args.eval_batch_size, collate_fn=collate,
drop_last = True
    )

    # multi-gpu evaluate
    if args.n_gpu > 1:
        model = torch.nn.DataParallel(model)

    # Eval!
    logger.info("***** Running evaluation {} *****".format(prefix))
    logger.info("  Num examples = %d", len(eval_dataset))
    logger.info("  Batch size = %d", args.eval_batch_size)
    eval_loss = 0.0
    nb_eval_steps = 0
    model.eval()

    for batch in tqdm(eval_dataloader, desc="Evaluating"):
        inputs, labels = (batch, batch)
        inputs = inputs.to(args.device)
        labels = labels.to(args.device)

        with torch.no_grad():
            outputs = model(inputs, labels=labels)
            lm_loss = outputs[0]
            eval_loss += lm_loss.mean().item()
        nb_eval_steps += 1

    eval_loss = eval_loss / nb_eval_steps
    perplexity = torch.exp(torch.tensor(eval_loss))

    result = {"perplexity": perplexity}
```

```
    output_eval_file = os.path.join(eval_output_dir, prefix, "eval_results.txt")
    with open(output_eval_file, "w") as writer:
        logger.info("***** Eval results {} *****".format(prefix))
        for key in sorted(result.keys()):
            logger.info("  %s = %s", key, str(result[key]))
            writer.write("%s = %s\n" % (key, str(result[key])))

    return result

# %%
# Main runner

def main(df_trn, df_val):
    args = Args()

    if args.should_continue:
        sorted_checkpoints = _sorted_checkpoints(args)
        if len(sorted_checkpoints) == 0:
            raise ValueError("Used --should_continue but no checkpoint was found in --output_dir.")
        else:
            args.model_name_or_path = sorted_checkpoints[-1]

    if (
        os.path.exists(args.output_dir)
        and os.listdir(args.output_dir)
        and args.do_train
        and not args.overwrite_output_dir
        and not args.should_continue
    ):
        raise ValueError(
            "Output directory ({}) already exists and is not empty. Use --overwrite_output_dir to
overcome.".format(
                args.output_dir
            )
        )

    # Setup CUDA, GPU & distributed training
    device = torch.device("cuda")
    args.n_gpu = torch.cuda.device_count()
    args.device = device

    # Setup logging
    logging.basicConfig(
        format="%(asctime)s - %(levelname)s - %(name)s -   %(message)s",
        datefmt="%m/%d/%Y %H:%M:%S",
        level=logging.INFO if args.local_rank in [-1, 0] else logging.WARN,
    )
    logger.warning(
        "Process rank: %s, device: %s, n_gpu: %s, distributed training: %s, 16-bits training: %s",
        args.local_rank,
        device,
        args.n_gpu,
        bool(args.local_rank != -1),
```

```
        args.fp16,
    )

    # Set seed
    set_seed(args)

    config = AutoConfig.from_pretrained(args.config_name, cache_dir=args.cache_dir)
    tokenizer = AutoTokenizer.from_pretrained(args.tokenizer_name, cache_dir=args.cache_dir)
    model = AutoModelWithLMHead.from_pretrained(
        args.model_name_or_path,
        from_tf=False,
        config=config,
        cache_dir=args.cache_dir,
    )
    model.to(args.device)

    logger.info("Training/evaluation parameters %s", args)

    # Training
    if args.do_train:
        train_dataset = load_and_cache_examples(args, tokenizer, df_trn, df_val, evaluate=False)

        global_step, tr_loss = train(args, train_dataset, model, tokenizer)
        logger.info(" global_step = %s, average loss = %s", global_step, tr_loss)

    # Saving best-practices: if you use save_pretrained for the model and tokenizer, you can reload
them using from_pretrained()
    if args.do_train:
        # Create output directory if needed
        os.makedirs(args.output_dir, exist_ok=True)

        logger.info("Saving model checkpoint to %s", args.output_dir)
        # Save a trained model, configuration and tokenizer using `save_pretrained()`.
        # They can then be reloaded using `from_pretrained()`
        model_to_save = (
            model.module if hasattr(model, "module") else model
        )  # Take care of distributed/parallel training
        model_to_save.save_pretrained(args.output_dir)
        tokenizer.save_pretrained(args.output_dir)

        # Good practice: save your training arguments together with the trained model
        torch.save(args, os.path.join(args.output_dir, "training_args.bin"))

        # Load a trained model and vocabulary that you have fine-tuned
        model = AutoModelWithLMHead.from_pretrained(args.output_dir)
        tokenizer = AutoTokenizer.from_pretrained(args.output_dir)
        model.to(args.device)

    # Evaluation
    results = {}
    if args.do_eval and args.local_rank in [-1, 0]:
        checkpoints = [args.output_dir]
        if args.eval_all_checkpoints:
```

```
            checkpoints = list(
                os.path.dirname(c) for c in sorted(glob.glob(args.output_dir + "/**/" +
WEIGHTS_NAME, recursive=True))
            )
            logging.getLogger("transformers.modeling_utils").setLevel(logging.WARN)  # Reduce
logging
        logger.info("Evaluate the following checkpoints: %s", checkpoints)
        for checkpoint in checkpoints:
            global_step = checkpoint.split("-")[-1] if len(checkpoints) > 1 else ""
            prefix = checkpoint.split("/")[-1] if checkpoint.find("checkpoint") != -1 else ""

            model = AutoModelWithLMHead.from_pretrained(checkpoint)
            model.to(args.device)
            result = evaluate(args, model, tokenizer, df_trn, df_val, prefix=prefix)
            result = dict((k + "_{}".format(global_step), v) for k, v in result.items())
            results.update(result)

    return results

# %% [markdown]
# ## Run the Main Function

# %%
main(trn_df, val_df)

# %% [markdown]
# ## Load the Trained Model

# %%
tokenizer = AutoTokenizer.from_pretrained('microsoft/DialoGPT-medium')
model = AutoModelWithLMHead.from_pretrained('output-small')

# %%
# Let's chat for 4 lines
for step in range(4):
    # encode the new user input, add the eos_token and return a tensor in Pytorch
    new_user_input_ids = tokenizer.encode(input(">> User:") + tokenizer.eos_token,
return_tensors='pt')
    # print(new_user_input_ids)

    # append the new user input tokens to the chat history
    bot_input_ids = torch.cat([chat_history_ids, new_user_input_ids], dim=-1) if step > 0 else
new_user_input_ids

    # generated a response while limiting the total chat history to 1000 tokens,
    chat_history_ids = model.generate(
        bot_input_ids, max_length=200,
        pad_token_id=tokenizer.eos_token_id,
        no_repeat_ngram_size=3,
        do_sample=True,
        top_k=100,
        top_p=0.7,
        temperature=0.8
```

```
    )

    # pretty print last ouput tokens from bot
    print("HarryBot: {}".format(tokenizer.decode(chat_history_ids[:, bot_input_ids.shape[-1]:][0],
skip_special_tokens=True)))

# %% [markdown]
# ## Push Model to Hugging Face

# %%
os.chdir('/content/')

# %%
!pip install huggingface_hub

# %%
!huggingface-cli login

# %%
!huggingface-cli repo create DialoGPT-medium-harrypotter

# %%
!sudo apt-get install git-lfs

# %%
!git clone https://huggingface.co/kozlovtsev/DialoGPT-medium-harrypotter

# %%
!ls "/content/drive/My Drive/output-small/"

# %%
!mv /content/drive/My\ Drive/output-small/* DialoGPT-medium-harrypotter/

# %%
os.chdir('DialoGPT-medium-harrypotter')

# %%
!git lfs install

# %%
!git add .

# %%
!git config --global user.email "dkozlovtsev1@gmail.com"
!git config --global user.name "Dmitry Kozlovtsev"

# %%
!git commit -m "Initial commit"

# %%
model.push_to_hub("DialoGPT-medium-harrypotter")

# %%
```

```
tokenizer.push_to_hub("DialoGPT-medium-harrypotter")

# %% [markdown]
# ## All Done!

# %%
```