**ПРАКТИЧНА РОБОТА №6**

**Мовні моделі**

Мета: Закріпити основні знання з використання великих мовних моделей для генерування текстів

Завдання:

Створити програму для автоматичного передбачення пропущених в тексті слів.

**Хід виконання.**

1. Інсталювати бібліотеку для LLM

pip install --upgrade transformers tensorflow

1. Імпортуємо необхідні бібліотеки

import tensorflow as tf

from tensorflow.keras import layers, models

from tensorflow.keras.datasets import mnist

import matplotlib.pyplot as plt

1. Завантажуємо токенайзер і модель

tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

model = TFBertForMaskedLM.from\_pretrained("bert-base-uncased")

*Якщо виникає помилка з завантаженням моделі, можливо у вас виник конфлікт з версією KERAS. Рішення:*

*Встановлення tf-keras: Виконайте наступну команду, щоб встановити tf-keras, який забезпечить зворотну сумісність:*

*pip install tf-keras*

*Видалення Keras: Якщо ви встановили Keras окремо, вам може знадобитися видалити його, щоб уникнути конфліктів:*

*pip uninstall keras*

*Оновлення TensorFlow: Переконайтеся, що у вас встановлена остання версія TensorFlow, оскільки нові версії Keras поставляються разом із TensorFlow. Ви можете оновити TensorFlow за допомогою:*

*pip install --upgrade tensorflow*

*Перезапуск середовища: Після внесення змін у бібліотеки, перезапустіть ваше середовище виконання або IDE, щоб оновлені зміни набрали чинності.*

1. Просимо користувача ввести текст з пропущеним словом на місці MASK



1. Підключаємо маску до моделі.

**tokenizer.mask\_token**: Це спеціальний токен, визначений токенізатором для використання в моделі BERT. Він використовується в процесі навчання та передбачення, щоб вказати моделі, що це місце потрібно заповнити.

**text.replace("[MASK]", tokenizer.mask\_token)**: Цей рядок замінює всі появи [MASK] у введеному тексті на реальний токен маски, який модель буде використовувати для виконання прогнозування. Наприклад, якщо токен маски дорівнює [MASK], то результатом буде текст, в якому замість [MASK] стоїть токен, зрозумілий моделі.

Таким чином, цей рядок коду готує текст для подальшої обробки моделлю, щоб вона могла правильно визначити, яке слово підходить на місце, позначене токеном маски.



1. Передаємо текст до токенізатору. Після виконання цього рядка, змінна inputs міститиме словник з тензорами (багатовимірними векторами), що представляють токени введеного тексту. Цей словник, як правило, містить такі ключі:
* **input\_ids**: Це тензор, що представляє індекси токенів (включаючи токен маски) у словнику токенізатора.
* **attention\_mask**: Це тензор, що вказує, які токени є частиною реального тексту (значення 1) та які токени є заповнювачами (значення 0).



1. Отримуємо індекси маски (місто на якому знаходиться маска)



1. Запускаємо модель



1. Отримаємо передбачення у вигляді токенів (перші 5)



1. Декодуємо їх у слова



1. Виводимо їх



**Спробуйте вивести не 5 а більше передбачених варіантів.**

Для такого невеликого набору даних неможливо оцінити якість моделі.

На цьому прикладі можна побачити як оцінюються показники якості (**спробуйте, зробіть висновки):**

# Імпорт необхідних бібліотек

import tensorflow as tf

from transformers import AutoTokenizer, TFBertForMaskedLM

from sklearn.metrics import accuracy\_score, precision\_score, recall\_score, f1\_score

# Загружаємо токенайзер і модель

tokenizer = AutoTokenizer.from\_pretrained("bert-base-uncased")

model = TFBertForMaskedLM.from\_pretrained("bert-base-uncased")

# Тестовий приклад

test\_texts = [

 "I love to eat [MASK] during the summer.",

 "The capital of France is [MASK].",

 "He is a famous [MASK]."

]

true\_values = ["ice cream", "Paris", "actor"] # Істинні значення

predictions = [] # Передбачені значення

for text in test\_texts:

 # Замінюємо '[MASK]' на маску моделі

 text\_with\_mask = text.replace("[MASK]", tokenizer.mask\_token)

 # Токенізація тексту

 inputs = tokenizer(text\_with\_mask, return\_tensors="tf")

 # Отримуємо індекси маски

 mask\_token\_index = tf.where(tf.equal(inputs["input\_ids"], tokenizer.mask\_token\_id))[0, 1]

 # Передбачення моделі

 outputs = model(\*\*inputs)

 logits = outputs.logits

 mask\_token\_logits = logits[0, mask\_token\_index, :]

 # Отримуємо топ-5 передбачень

 top\_5\_tokens = tf.math.top\_k(mask\_token\_logits, 5).indices.numpy()

 # Декодуємо передбачені токени в слова

 predicted\_words = [tokenizer.decode([token], skip\_special\_tokens=True).strip() for token in top\_5\_tokens]

 # Зберігаємо перше передбачене слово

 predictions.append(predicted\_words[0]) # Додаємо лише перше передбачене слово

# Вивід результатів

for i, (true\_value, prediction) in enumerate(zip(true\_values, predictions)):

 print(f"Текст {i+1}:")

 print(f"Істинне слово: {true\_value}")

 print(f"Передбачене слово: {prediction}")

 print()

# Обчислення метрик

accuracy = accuracy\_score(true\_values, predictions)

precision = precision\_score(true\_values, predictions, average='weighted', zero\_division=0)

recall = recall\_score(true\_values, predictions, average='weighted', zero\_division=0)

f1 = f1\_score(true\_values, predictions, average='weighted', zero\_division=0)

# Вивід метрик

print("\nМетрики якості:")

print(f"Точність (Accuracy): {accuracy \* 100:.2f}%")

print(f"Precision: {precision:.2f}")

print(f"Recall: {recall:.2f}")

print(f"F1-Score: {f1:.2f}")

Для покращення роботи моделі можна спробувати експериментувати з архітектурою LLM.

* 1. **Вибір моделі:**

Модель BERT має кілька варіантів з різними розмірами, які ви можете використовувати для експериментів:

* 1. **BERT-base (12 шарів, 12 голов уваги, 110 млн параметрів)**
	2. **BERT-large (24 шари, 16 голов уваги, 340 млн параметрів)**
	3. **Інші моделі:** Ви можете експериментувати з варіантами BERT, такими як RoBERTa, ALBERT або DistilBERT, які відрізняються архітектурними рішеннями і розмірами.

Наприклад:

tokenizer = AutoTokenizer.from\_pretrained("bert-large-uncased") model = TFBertForMaskedLM.from\_pretrained("bert-large-uncased")

2. **Кількість голів уваги (Number of Attention Heads):**

Модель BERT використовує **механізм багатоголової уваги**. Кожна "голова" уваги дозволяє моделі зосереджуватися на різних частинах вхідного тексту.

Ви можете експериментувати зі зміною кількості голів уваги:

* Модель BERT-base використовує 12 голів уваги, тоді як BERT-large використовує 16.
* Збільшення кількості голів може покращити здатність моделі враховувати контекст, але збільшить обчислювальні витрати.

**3.Розмір прихованих шарів (Hidden Size):**

**Розмір прихованих шарів** визначає кількість нейронів на кожному шарі трансформера.

* Для BERT-base це 768 нейронів, а для BERT-large — 1024.

Ви можете досліджувати моделі з меншим або більшим розміром прихованих шарів, що впливає на здатність моделі захоплювати складну інформацію з тексту

* 1. **Dropout:**

**Dropout** — це метод регуляризації, який допомагає уникнути перенавчання моделі, вимикаючи випадкові нейрони під час навчання. Можна експериментувати зі значенням Dropout, щоб покращити узагальнення моделі:

В стандартній BERT використовується 10% Dropout (0.1).

Можна збільшити його до 0.3 або 0.5 для складніших завдань або зменшити для завдань з малою кількістю даних:

from transformers import TFBertConfig, TFBertForMaskedLM

# Кастомна конфігурація

config = TFBertConfig(

 hidden\_size=512, # Розмір прихованих шарів

 num\_hidden\_layers=6, # Кількість шарів

 num\_attention\_heads=8, # Кількість голів уваги

 intermediate\_size=2048, # Розмір проміжних шарів

 hidden\_dropout\_prob=0.2 # Ймовірність dropout

)

# Створення моделі з кастомною конфігурацією

model = TFBertForMaskedLM(config)

# Виведення архітектури моделі

model.summary()

Спробуйте застосувати деякі з цих методів, але

**Рекомендації:**

1. **Поступові зміни:** Робіть зміни в архітектурі поступово, щоб краще розуміти їхній вплив на якість моделі.
2. **Тестування на реальних даних:** Експериментуйте з моделлю на конкретних реальних даних для того, щоб переконатися, що зміни позитивно впливають на продуктивність.