Data and Artificial Intelligence

Cyber Shujaa Program

Week 11 Assignment
Natural Language processing with Transformers

Student Name: Deborah Kwamboka Omae

Student ID: CS-DA02-25075

Table of Contents

Table of figures

Introduction

This assignment involved implementing my understanding of Natural Language Processing. NLP focuses on enabling computers to understand, interpret and generate human language in ways that are both meaningful and useful. It has two subfields. Natural Language Understanding (NLU) which focuses on semantic analysis or determining the intended meaning of text and Natural Language Generation (NLG) which focuses on generating meaningful text.

Objectives

- Apply a pre-trained BERT model for sentence encoding using Hugging Face Transformers.
- Extract token-level contextual embeddings.
- Demonstrate how BERT captures word meaning based on context (polysemy).
- Compute and interpret cosine similarity between token embeddings.
- Explain the importance of contextual embeddings in contrast to static word vectors.

Tasks Completed

Step 1: importing the necessary libraries

In this step, I imported the necessary libraries that will be used and loading the pre-trained models

Code:

```
## Import required libraries

from transformers import BertTokenizer, TFBertModel

import tensorflow as tf

import numpy as np

from sklearn.metrics.pairwise import cosine_similarity
```

 # Load pre-trained BERT tokenizer and BERT model from Hugging Face

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

bert_model = TFBertModel.from_pretrained('bert-base-uncased')

code explanation

BERT Tokenizer converts text into tokens (subwords) that BERT understands and BERT Model is the actual neural network that processes tokens into embeddings. Numpy is for numerical computations and cosine similarity is used to compare vectors.

Screenshot:

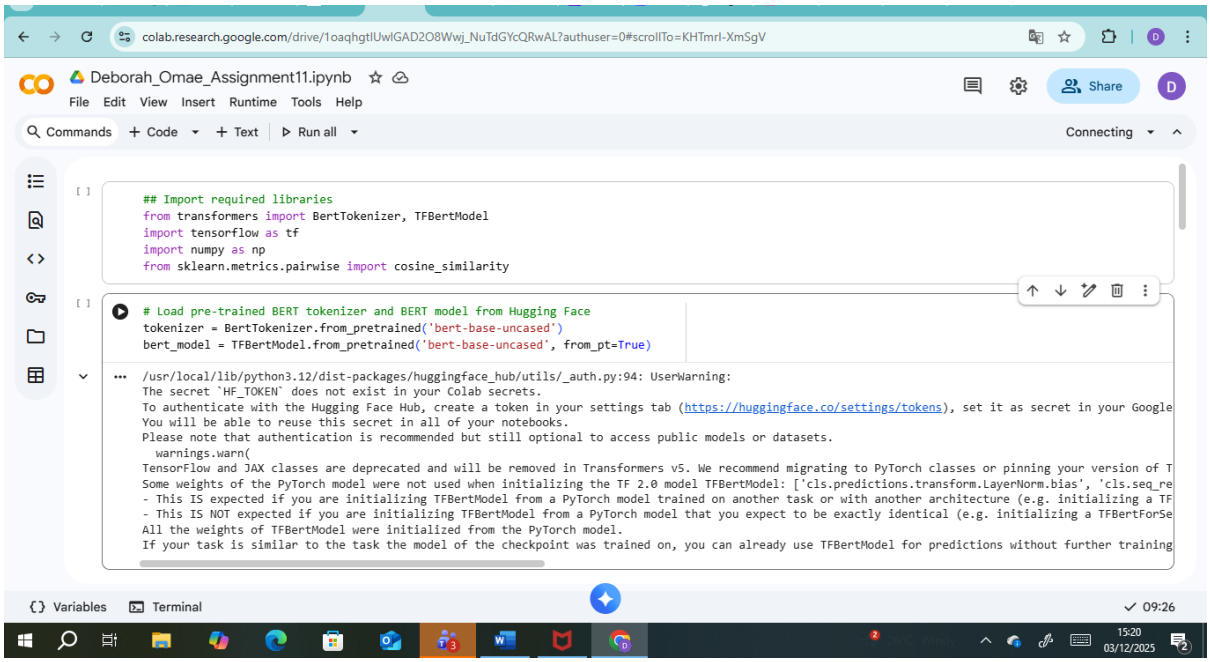

Figure 1: screenshot showing code for importing libraries and loading the model

Step 2: Data preparation

In this step I created 10 sentence pairs which had to be evaluated

Code:

 # Example sentence pairs (some similar, some not)

sentence_pairs = [

("How do I learn Python?", "What is the best way to study Python?"),

("What is AI?", "How to cook pasta?"),

("How do I bake a chocolate cake?", "Give me a chocolate cake recipe."),

("How can I improve my coding skills?", "Tips for becoming better at programming."),

("Where can I buy cheap laptops?", "Best sites to find affordable computers."),

("Is marriage a scam?", "Negative views on marriage."),

("What is the importance of Aloe Vera?", "Benefits of Aloe Vera plant."),

("Why do I have to eat healthy?", "Reasons for fasting."),

("Is unemployment frustrating?", "The downside of unemployment."),

("How can I make more friends?", "Tips on attracting friendships."),

]

 # Ground truth similarity labels: 1 = similar, 0 = not similar

labels = [1, 0, 1, 1, 1, 1, 1, 0, 1, 1]

screenshot:

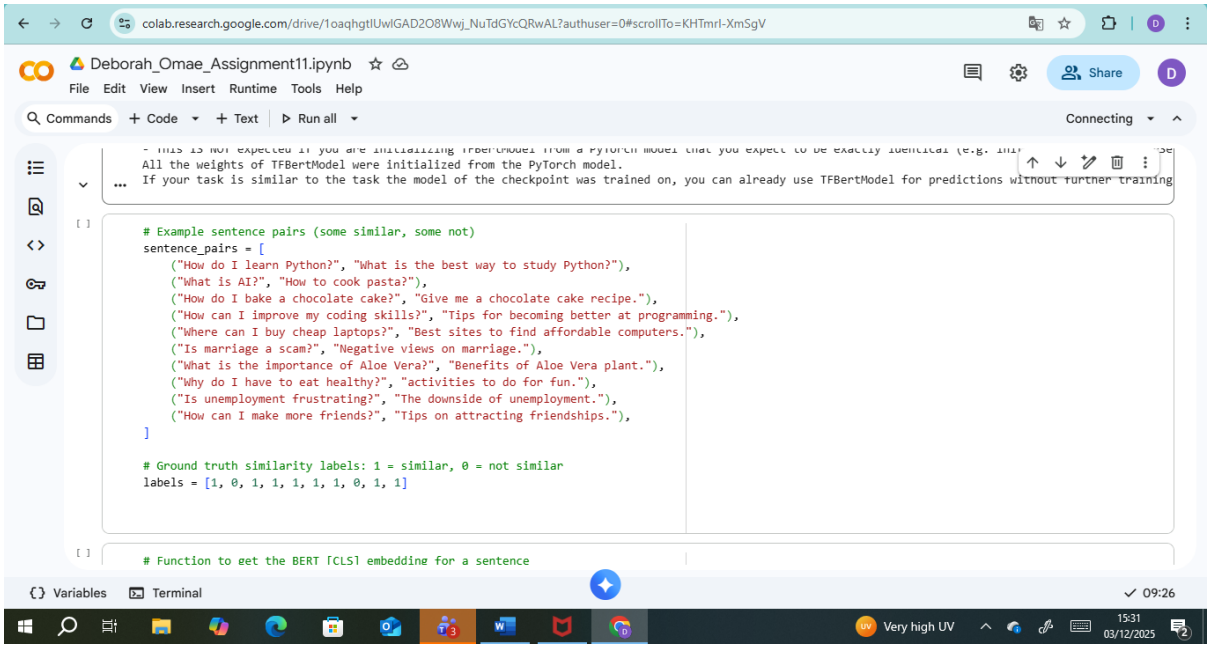

Figure 2: screenshot showing the created sentence pairs

Step 3: Getting embedding for the sentence

In this step I create a function that takes raw text and returns a 768-dimensional vector representing the sentence's meaning.

Code:

```
# Function to get the BERT [CLS] embedding for a sentence

def get_sentence_embedding(sentence):

    # Tokenize and encode sentence into input tensors

    inputs = tokenizer(sentence, return_tensors='tf', padding=True, truncation=True)

    # Get model output

    outputs = bert_model(inputs)

    # Extract [CLS] token embedding (shape: [1, 768])

    cls_embedding = outputs.last_hidden_state[:, 0, :]

    return cls_embedding.numpy()
```

screenshot:

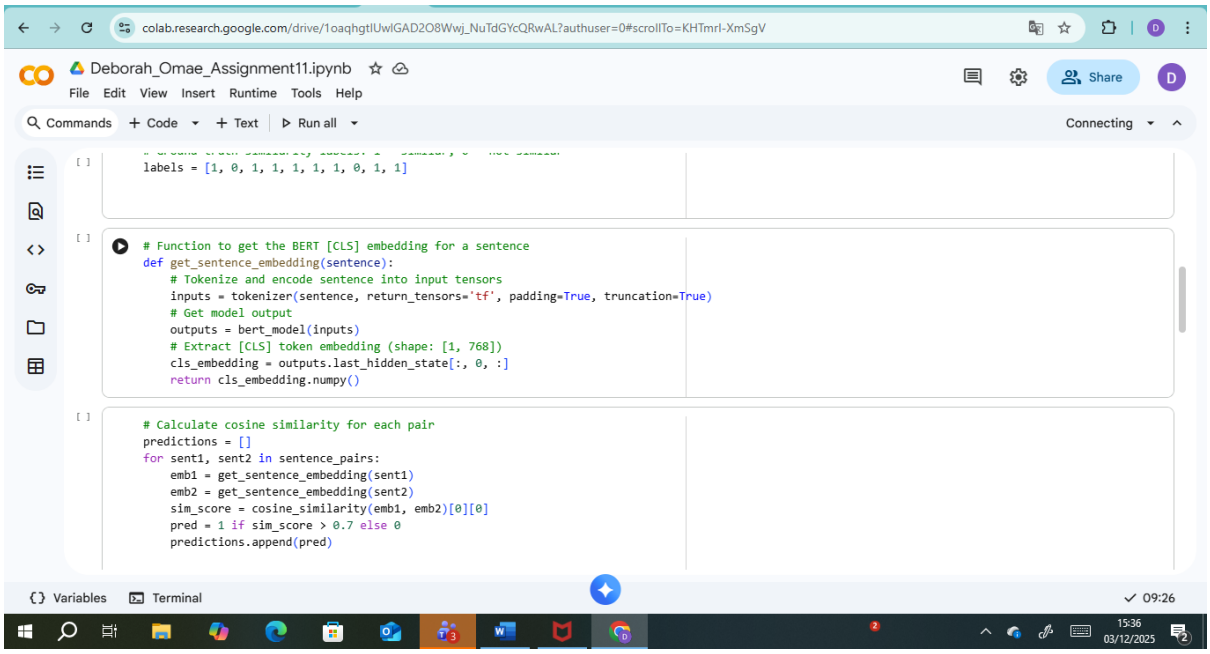

Figure 3: screenshot showing the code for embedding function

Step 4: cosine similarity

In this step I computed the cosine similarity of the vectors of the sentence pairs

Code:

```
# Calculate cosine similarity for each pair

predictions = []

for sent1, sent2 in sentence_pairs:

    emb1 = get_sentence_embedding(sent1)

    emb2 = get_sentence_embedding(sent2)

    sim_score = cosine_similarity(emb1, emb2)[0][0]

    pred = 1 if sim_score > 0.7 else 0

    predictions.append(pred)



    print(f"\nSentence 1: {sent1}")
```

print(f"Sentence 2: {sent2}")

print(f"Cosine Similarity: {sim_score:.4f} → Predicted Similar: {pred}")

screenshot:

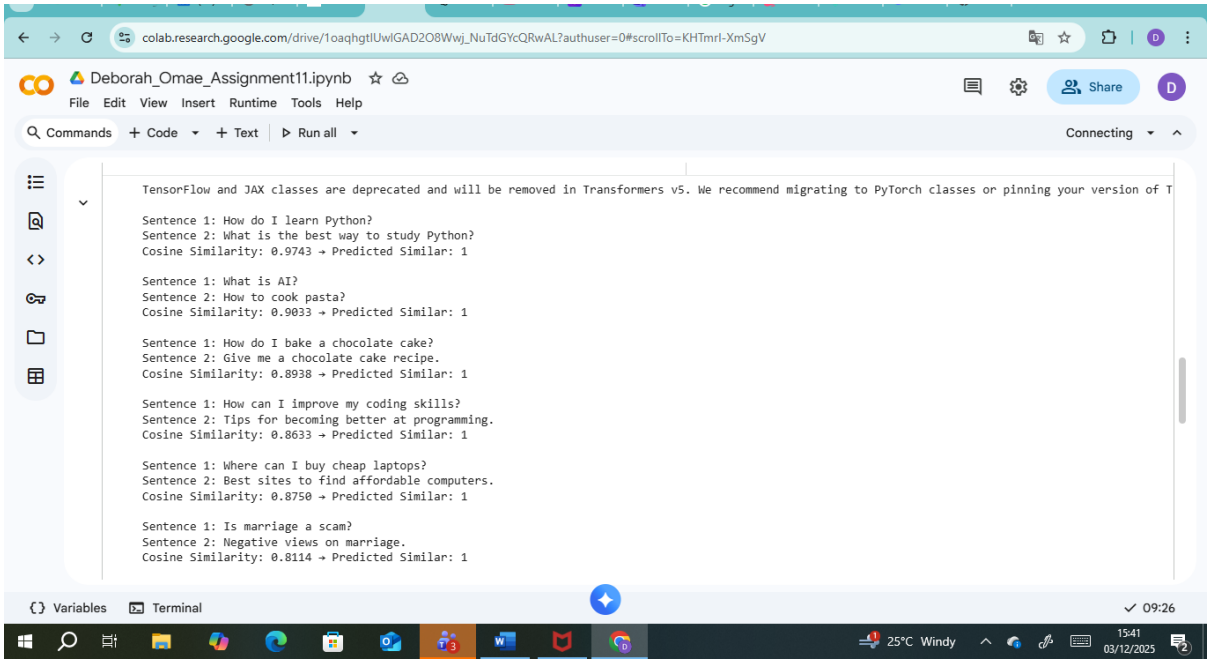

Figure 4: screenshot showing the result of calculating cosine similarity

Step 5: accuracy evaluation

In this step I evaluated the accuracy of my BERT model. I had created the variable labels which had stored the similarity score by manually evaluating.

Code:

 # Evaluate accuracy

correct = 0

for i in range(len(predictions)):

if predictions[i] == labels[i]:

correct += 1

 # Final accuracy calculation

total = len(labels)

accuracy = correct / total

print(f"\nAccuracy: {accuracy:.2%}")

screenshot:

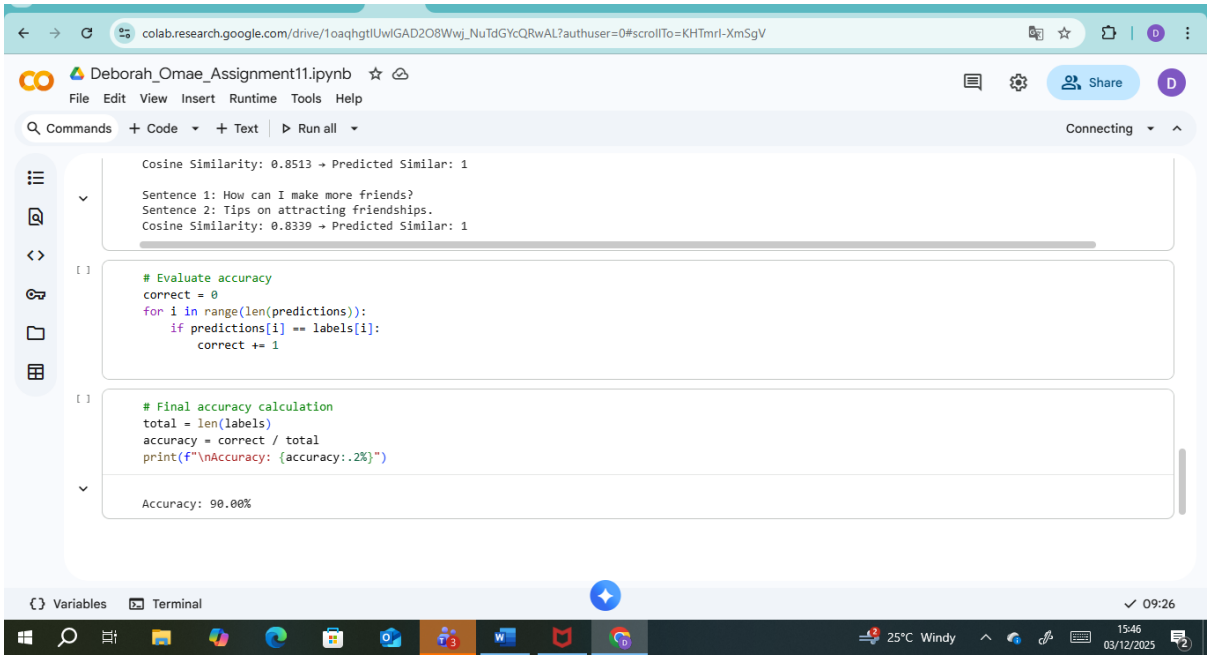

Figure 5: screenshot showing accuracy evaluation

Demonstrate how BERT captures word meanings based on context (polysemy)

BERT handles polysemy (words with multiple meanings) by using contextual embeddings — meaning the representation of a word changes depending on the words around it.

For example, in the sentence *"He sat by the bank of the river,"* BERT looks at surrounding words like *river* and produces an embedding that reflects the geographical meaning of *bank*. But in the sentence *"She deposited money in the bank,"* BERT sees words like *money* and *deposited* and generates a completely different embedding that represents a financial institution. Even though the word is spelled the same, BERT's contextual

embeddings shift based on nearby words, allowing it to correctly interpret the intended meaning.

Explain the importance of contextual embeddings in contrast to static word vectors.

Contextual embeddings are important because they allow models like BERT to understand what a word means in a specific sentence, unlike static word vectors that assign one fixed meaning to a word everywhere. This leads to much better performance in tasks such as sentiment analysis, question answering, translation, and anything requiring true language understanding

Link to Code

Link to Code:

https://colab.research.google.com/drive/1oaqhgtIUwlGAD2O8Wwj_NuTdGYcQRwAL?usp=sharing

Conclusion

In this assignment, I explored how BERT can be used to measure sentence similarity by generating contextual embeddings for each sentence and comparing them using cosine similarity. Unlike traditional models with static word vectors, BERT provides meaningful sentence-level representations that capture the true intent and context behind the text. By computing cosine similarity scores for different sentence pairs, I was able to classify whether the sentences were semantically similar or not, and evaluate the accuracy of the model's predictions. Through this task, I learned how to load and use a pre-trained BERT model in Python, extract [CLS] embeddings, compute similarity metrics, and interpret the results. Overall, the assignment helped me understand the power of contextual embeddings and how BERT can be applied to real NLP tasks such as semantic similarity and text understanding.