

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

Data and Artificial Intelligence	1
Cyber Shujaa Program.....	1
Week 11 Assignment Natural Language processing with Transformers	1
Introduction	4
Objectives.....	4
Tasks Completed	4
Step 1: importing the necessary libraries	4
Step 2: Data preparation	5
Step 3: Getting embedding for the sentence	7
Step 4: cosine similarity	8
Step 5: accuracy evaluation	9
Demonstrate how BERT captures word meanings based on context (polysemy)	10
Explain the importance of contextual embeddings in contrast to static word vectors.	11
Link to Code	11
Conclusion.....	11

Table of figures

Figure 1: screenshot showing code for importing libraries and loading the model	5
Figure 2: screenshot showing the created sentence pairs.....	7
Figure 3: screenshot showing the code for embedding function	8
Figure 4: screenshot showing the result of calculating cosine similarity.....	9
Figure 5: screenshot showing accuracy evaluation.....	10

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

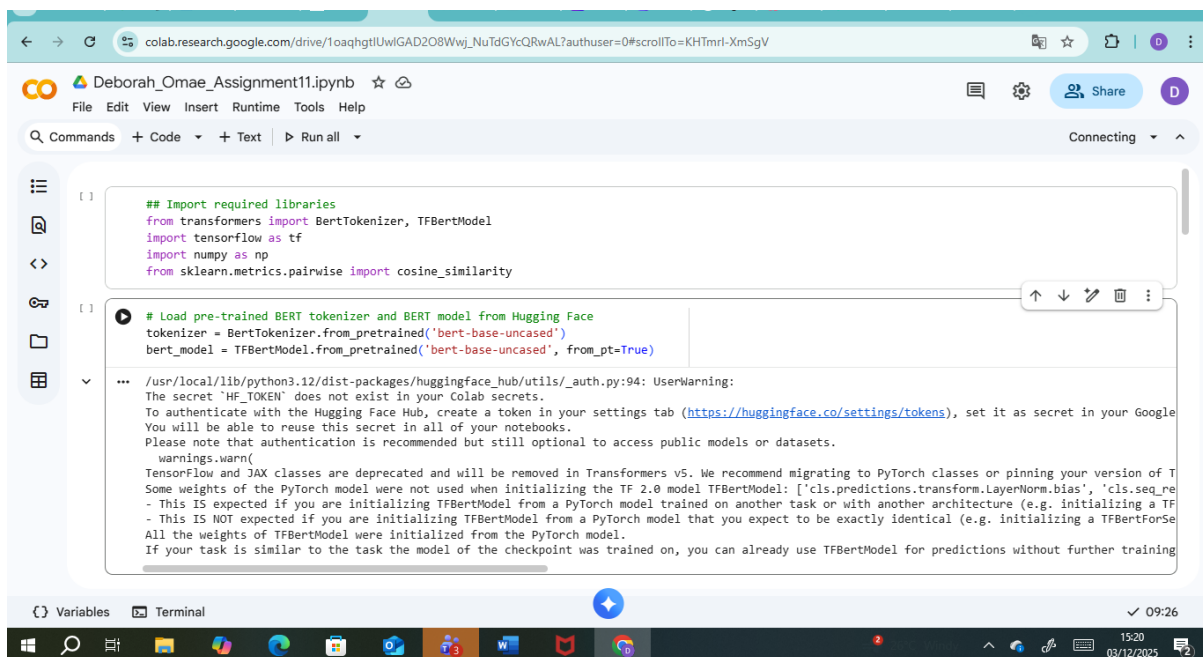
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

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:



```

## 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', from_pt=True)
  
```

... /usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret 'HF_TOKEN' does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (<https://huggingface.co/settings/tokens>), set it as secret in your Google
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
TensorFlow and JAX classes are deprecated and will be removed in Transformers v5. We recommend migrating to PyTorch classes or pinning your version of T
Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFBertModel: ['cls.predictions.transform.LayerNorm.bias', 'cls.seq_re
- This IS expected if you are initializing TFBertModel from a PyTorch model trained on another task or with another architecture (e.g. initializing a TF
- This IS NOT expected if you are initializing TFBertModel from a PyTorch model that you expect to be exactly identical (e.g. initializing a TFBertForSe
All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without further training

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:

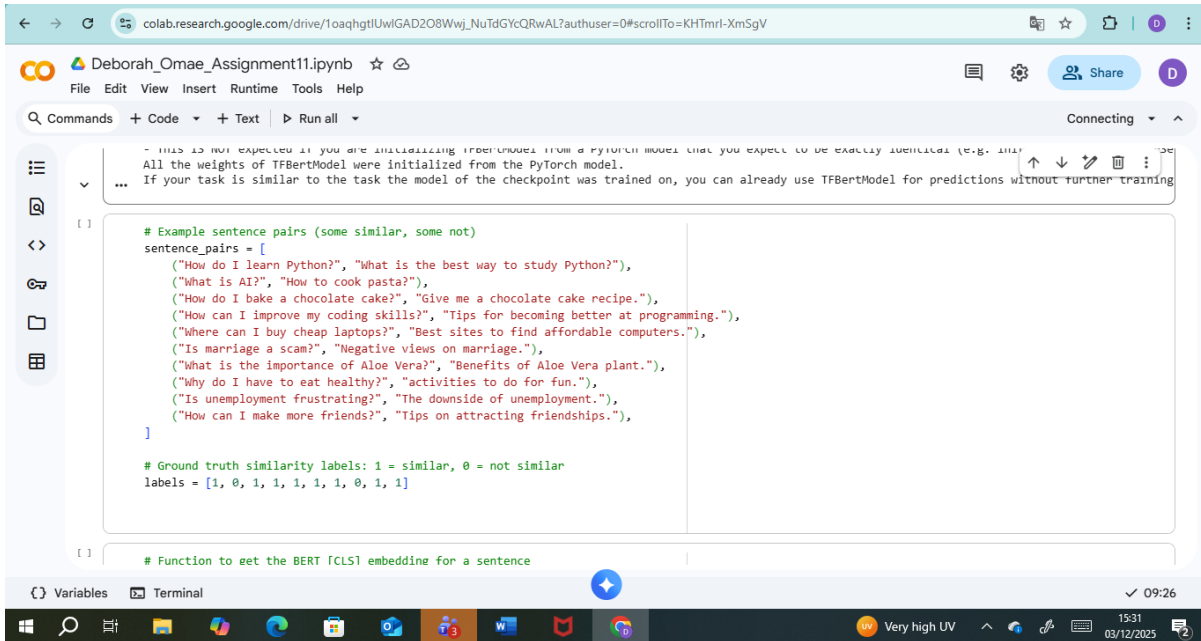
Example sentence pairs (some similar, some not)

```
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:



```

- This is not expected if you are initializing from a PyTorch model that you expect to be exactly identical (e.g., init
... All the weights of TFBertModel were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFBertModel for predictions without further training

# 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?", "activities to do for fun."),
    ("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]

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

```

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

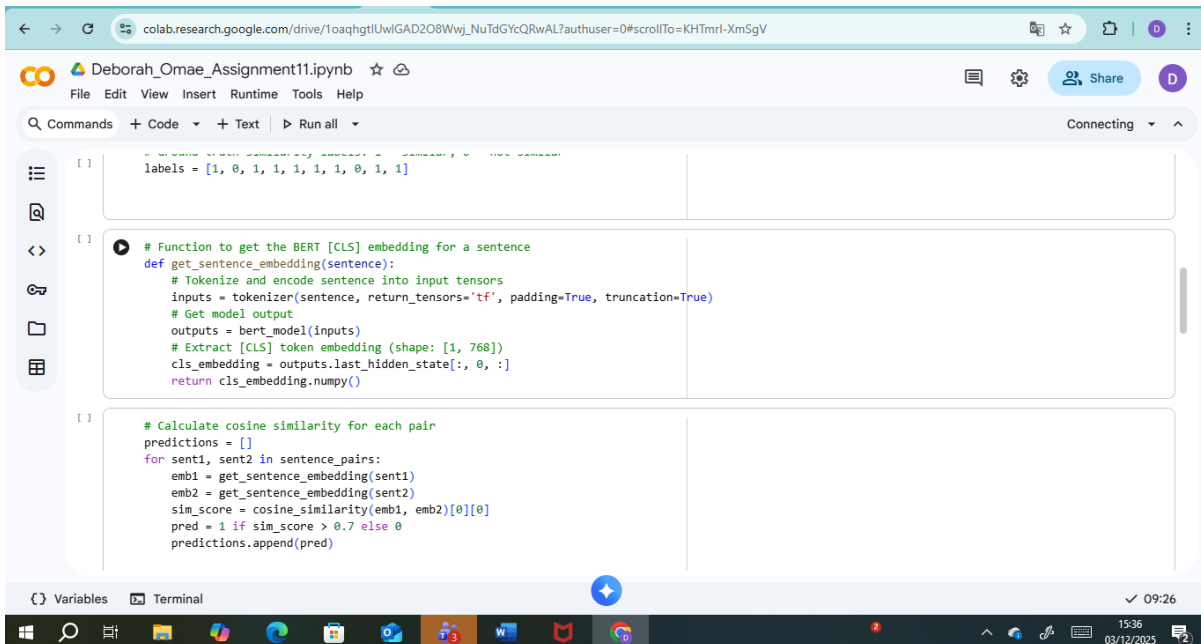
outputs = bert_model(inputs)

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

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

return cls_embedding.numpy()

screenshot:



```

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

[ ] # 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()

[ ] # 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)

```

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:

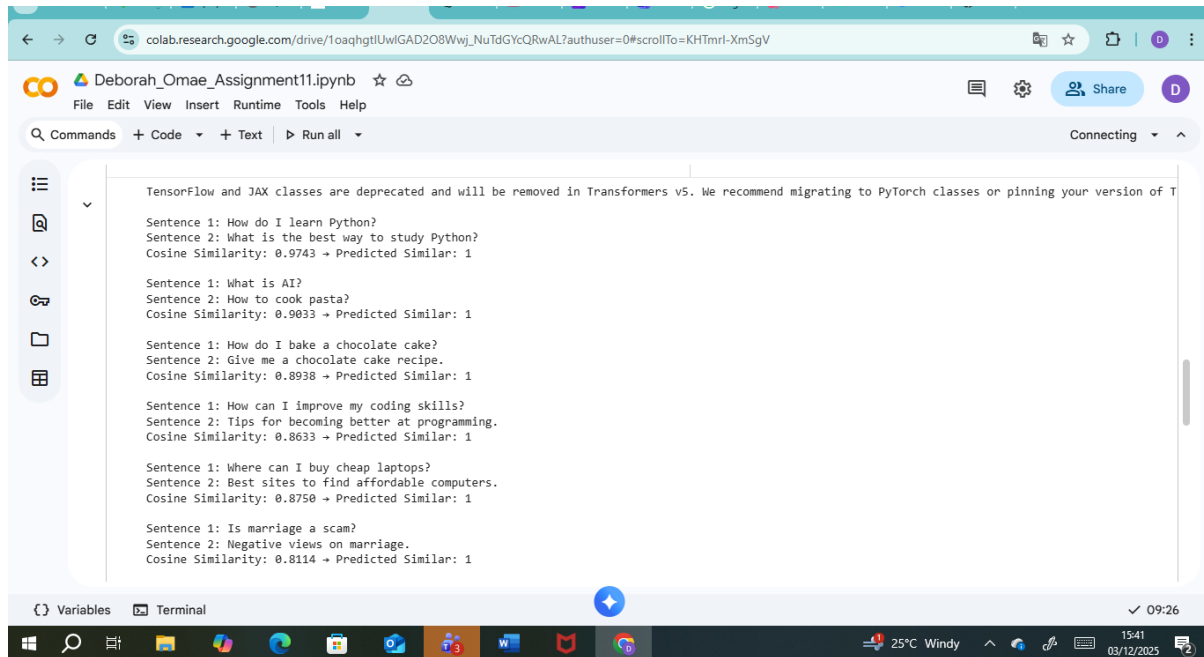


Figure 4: screenshot showing the result of calculating cosine similarity

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
```

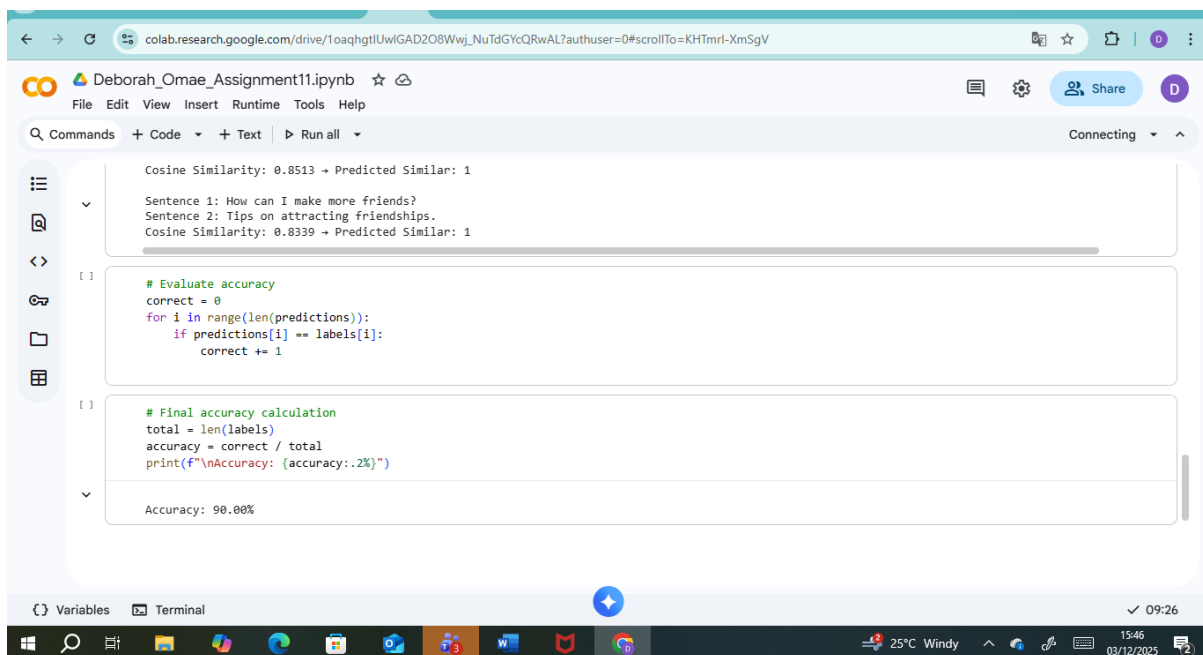
```
# Final accuracy calculation
```

```
total = len(labels)
```

```
accuracy = correct / total
```

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

screenshot:

A screenshot of a Google Colab notebook titled 'Deborah_Omae_Assignment11.ipynb'. The notebook interface shows a menu bar with 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with 'Commands', '+ Code', '+ Text', and 'Run all'. The main area contains three code cells. The first cell shows cosine similarity calculations for two sentences. The second cell evaluates accuracy by comparing predictions to labels. The third cell calculates the final accuracy, which is displayed as 'Accuracy: 90.00%'. The bottom of the screen shows a Windows taskbar with various application icons and a system tray with weather and time information.

```
Cosine Similarity: 0.8513 -> Predicted Similar: 1
Sentence 1: How can I make more friends?
Sentence 2: Tips on attracting friendships.
Cosine Similarity: 0.8339 -> Predicted Similar: 1

[ ]
# 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%}")

Accuracy: 90.00%
```

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/1oaqhgtIUwIGAD2O8Wwj_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.