CS 4501 Natural Language Processing (Fall 2024)
University of Virginia
Instructor: Yu Meng
Assignment prepared by TAs: Zhepei Wei & Wenqian Ye

Assignment 4: Transformer Language Model (132 points)

Davis Wang

0. *Setups and Instructions* (0 pts).
   In this assignment, we will primarily use this **Jupyter Notebook** to complete the tasks.

   Ensure that all the required packages are installed before you begin. You can either use Google Colab or run Jupyter Notebook on your local machine. For running Jupyter Notebook locally, refer to this guide to setup your own machine.

   Output Requirement: You should paste **your code (only your completed parts or functions)** AND **the results/plots** into the corresponding `verbatim` cells in this file.

1. *Tokenization with BertTokenizer.* (25 pts)

   (a) Use BertTokenizer to segment the given text sequences. (10 pts)

```
# Tokenize the input text
texts = ["Transformers are powerful tools in natural language processing.",
         "Tokenization is a crucial step in NLP."]

for text in texts:
  # TODO: tokenize the text using tokenizer
  tokens = tokenizer.tokenize(text)

  print("Tokens:", tokens)

  Tokens: ['transformers', 'are', 'powerful', 'tools', 'in', 'natural',
  'language', 'processing', '.']
Tokens: ['token', '##ization', 'is', 'a', 'crucial', 'step', 'in', 'nl',
'##p', '.']
```

   (b) Count how many subword tokens, starting with "##", are there in BERT's vocabulary). (5 pts)

```
subword_count = 0
for token in tokenizer.vocab:
    if token.startswith("##"):
      subword_count+=1
```

```
print("Number of subword tokens starting with '##':", subword_count)

Number of subword tokens starting with '##': 5828
```

(c) In two sentences, explain why some subword tokens start with "##" while others do not, and why is subword tokenization beneficial in language models? (10 pts)

Some tokens are formatted with "\#\#" to indicate that they are continuations of previous tokens. This type of tokenization is beneficial because it gives models a practical way of handling rare words and common words by finding a suitable balance between vocabulary size and coverage.

2. Use BERT representations for word sense disambiguation. (35 pts)

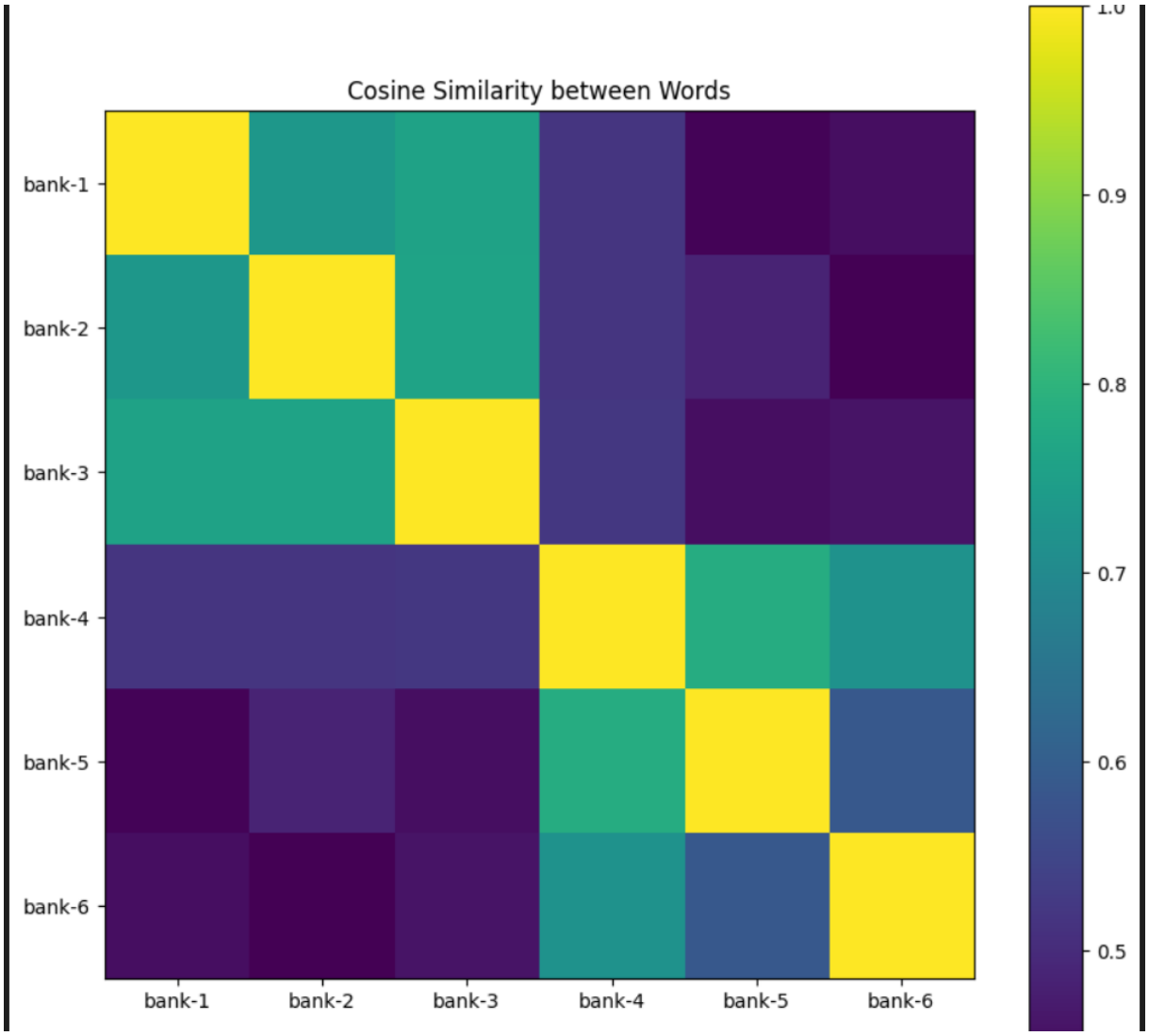
(a) Given the sentences with polysemous words, use BERT's last layer representation to compute the cosine similarity between same-meaning polysemy occurrences and different-meaning occurrences. (10 pts)

```
# Compute cosine similarity
num_sentences = len(sentences)
similarity_matrix = np.zeros((num_sentences, num_sentences))
for i in range(num_sentences):
  for j in range(num_sentences):
    vec1 = representations[i]
    vec2 = representations[j]

    similarity = None
    # TODO: compute cosine similarity between vec1 and vec2
    similarity = np.dot(vec1, vec2) / (np.linalg.norm(vec1) * np.linalg.norm(ve

    similarity_matrix[i, j] = similarity

Cosine similarity matrix:
[[1.         0.73256558 0.75830871 0.51688361 0.43581358 0.45084441]
 [0.73256558 0.99999994 0.76052368 0.5171715  0.48687994 0.42950669]
 [0.75830871 0.76052368 1.         0.52252835 0.45136365 0.46047783]
 [0.51688361 0.5171715  0.52252835 1.         0.78341585 0.71935844]
 [0.43581358 0.48687994 0.45136365 0.78341585 1.         0.58699131]
 [0.45084441 0.42950669 0.46047783 0.71935844 0.58699131 1.        ]]
```

(b) Plot the pairwise cosine similarity of different polysemy occurrences. (5 pts)

```
# TODO: visualize similarity matrix
plt.figure(figsize=(10, 10))
plt.imshow(similarity_matrix, interpolation='nearest')


plt.colorbar()
```

```
plt.xticks(np.arange(len(sentences)), [f'bank-{i+1}' for i in range(len(sentenc
plt.yticks(np.arange(len(sentences)), [f'bank-{i+1}' for i in range(len(sentenc
plt.title("Cosine Similarity between Words")
plt.show()
```



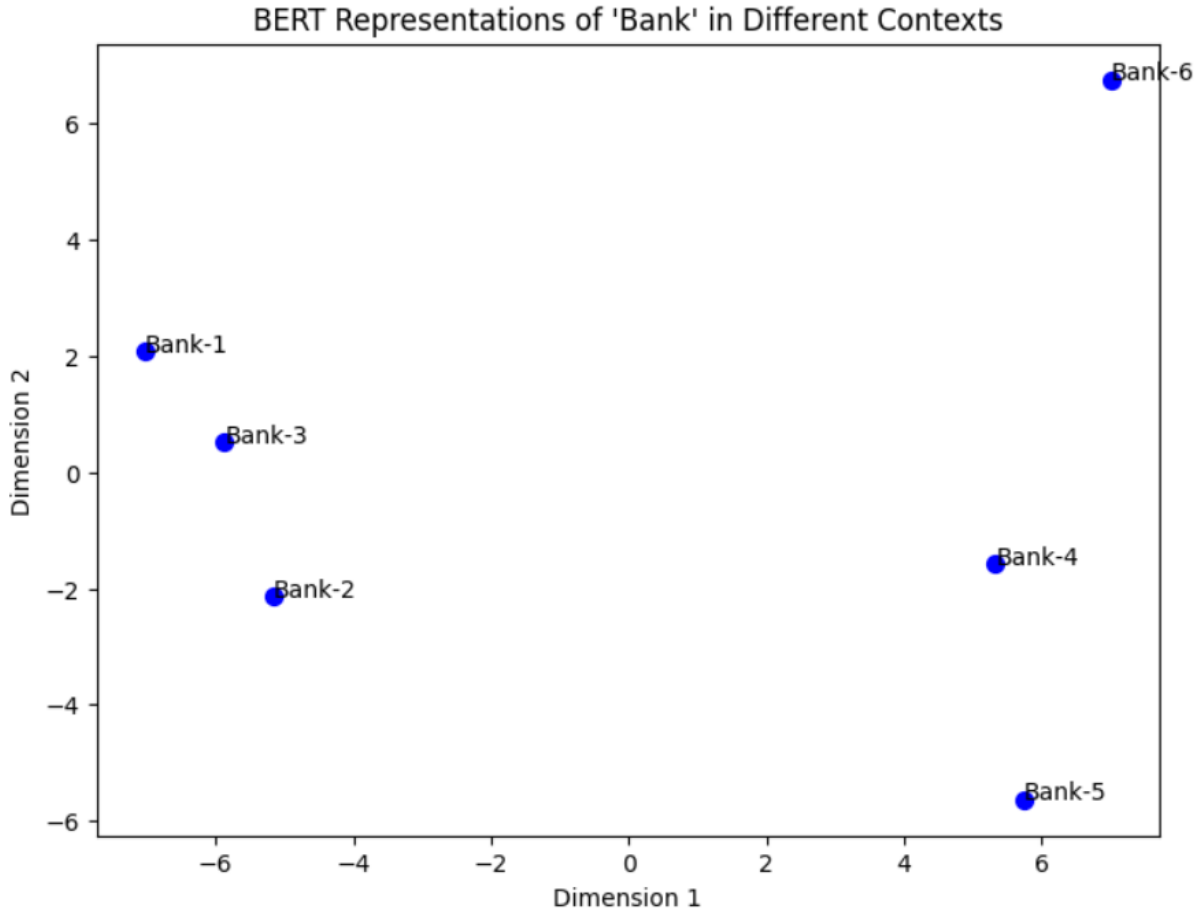
(c) In one sentence, describe how the plot in (b) would look like if we used Word2Vec embeddings to compute the pairwise cosine similarity of different polysemy occurrences. (5 pts)

```
If we used word embeddings to compute pairwise cosine similarity between differ
```

(d) Use principal component analysis (PCA) to reduce the dimension of these representations and visualize them using a 2D scatter plot. (10 pts)

```
# Visualize the representations
plt.figure(figsize=(8, 6))
for i, representation in enumerate(reduced_representations):
  # TODO: use plt.scatter() to draw the scatter plot
  plt.scatter(representation[0], representation[1], c='blue', s=50)

  # add text to each point in the plot
  plt.text(representation[0], representation[1], f"Bank-{i + 1}")
```

```
plt.xlabel("Dimension 1")
plt.ylabel("Dimension 2")
plt.title("BERT Representations of 'Bank' in Different Contexts")
plt.show()
```



(e) In one sentence, analyze the plots in (b) and (d). (5 pts)

The plots in (b) and (d) are corresponding representations of how different usage cases of the word 'bank' fall under different categories, where the plot in (b) shows a one-to-one relationship between individual contextual representations of 'bank', and plot (d) organizes each contextual representation in clusters for sentences that utilize similar meanings for 'bank'.

3. Prompt BERT for NLP tasks. (36 pts)

(a) Implement the masked word prediction function (10 pts)

```
# Use BERT to predict masked words
def predict_masked_words(sentences, model):
  for idx, sentence in enumerate(sentences):
      # TODO: use loaded pipeline model to complete the sentence
      predictions = model(sentence)

      print(f"Prediction {idx}:\n{predictions[0]}\n")
```

(b) Topic Classification (e.g., "Machine learning is a subset of [MASK].", where [MASK] is the topic word.) (6 pts)

```
Prediction 0:
{'score': 0.3448408246040344, 'token': 6622, 'token_str': 'evolution',
'sequence': 'the theory of evolution and natural selection is about evolution.'

Prediction 1:
{'score': 0.21643176674842834, 'token': 7072, 'token_str': 'democracy',
'sequence': 'democracy and governance fall under the study of democracy.'}

Prediction 2:
{'score': 0.5235655903816223, 'token': 5543, 'token_str': 'economics',
'sequence': 'the concept of supply and demand is fundamental to economics.'}
```

(c) Sentiment Classification (e.g., "I really enjoyed this movie, it is [MASK].", where [MASK] is the sentiment label.) (6 pts)

```
Prediction 0:
{'score': 0.6532092094421387, 'token': 3866, 'token_str': 'loved',
'sequence': 'i loved buying this item, it was a wonderful experience.'}

Prediction 1:
{'score': 0.06757467985153198, 'token': 9479, 'token_str': 'lonely',
'sequence': 'the weather today is gloomy, making me feel lonely.'}

Prediction 2:
{'score': 0.14031945168972015, 'token': 6429, 'token_str': 'amazing',
'sequence': 'the movie was absolutely fantastic, it was amazing.'}
```

(d) In one sentence, propose one method that converts the predicted words in (b) and (c) into categorical predictions (e.g., if we obtain "amazing"/"great"/"terrible"/"awful" etc. as the predicted word for a sentence, how can we know it is positive or negative sentiment?) (8 pts)

```
One method we could try is to implement a predetermined set of mappings for
words that are either categorized as positive or negative in sentiment, which
would allow us to perform categorical predications for matching or similar
word tokens.
```

(e) Factual Knowledge Probing (e.g., "The capital of France is [MASK].", where [MASK] is the answer.) (6 pts)

```
rediction 0:
{'score': 0.416788786649704, 'token': 3000, 'token_str': 'paris',
'sequence': 'the capital of france is paris.'}

Prediction 1:
{'score': 0.28096118569374084, 'token': 2859, 'token_str': 'china',
'sequence': 'the great wall of china is a famous historical structure.'}

Prediction 2:
```

```
{'score': 0.04522893577814102, 'token': 13963, 'token_str': '1776',
'sequence': 'the united states declared independence in the year 1776.'}
```

4. Fine-tune BERT for text classification. (36 pts)

   (a) Use the provided training data to fine-tune a BERT model on a binary sentiment classification task. (15 pts)

```
eval_log = None
# TODO: fine-tune and evaluate the BERT model using trainer.train() and trainer
trainer.train()
eval_log = trainer.evaluate()

print(f'Evaluation Log: {eval_log}')

Evaluation Log: {'eval_loss': 0.3163357973098755, 'eval_runtime': 5.4037, 'eval
```

   (b) Evaluate the fine-tuned BERT model on the given test set and report the test accuracy. (15 pts)

```
# Iterate through the test dataset
for i in range(total_predictions):
  label = test_data[i]['label']
  inputs = tokenizer([test_data[i]['text']], padding=True, truncation=True, ret

  # TODO (for problem 4(b)): Use the fine-tuned model to get a prediction for t
  outputs = model(**inputs)

  # Apply softmax to get probabilities
  probabilities = torch.softmax(outputs.logits, dim=1)
  prediction = torch.argmax(probabilities, dim=1)

  if prediction == label:
    correct_predictions += 1
  # TODO (for problem 4(c)): collect sentences from the test set for which the
  else:
    print(f"Incorrect Prediction:")
    print(f"Text: {test_data[i]['text'][:200]}...")  # Print first 200 char
    print(f"True label: {label}")
    print(f"Predicted label: {prediction}")
    print("-" * 50)  # Print a separator for readability

Total correct predictions: 178
Total predictions: 200
Test Accuracy: 0.89
```

(c) List three sentences from the test set for which the fine-tuned BERT model made wrong predictions. (6 pts)

```
--------------------------------------------------
Incorrect Prediction:
Text: The barbarians maybe´s not the best film that anybody of us have seen,
but really????........It´s so funny......I can´t discribe how mutch I laughed
when I first saw it..The director really wanted to ...
True label: 1
Predicted label: tensor([0], device='cuda:0')
--------------------------------------------------
Incorrect Prediction:
Text: Hard to believe, perhaps, but this film was denounced as immoral from
more pulpits than any other film produced prior to the imposition of the
bluenose Hayes Code. Yes indeed, priests actually told th...
True label: 1
Predicted label: tensor([0], device='cuda:0')
--------------------------------------------------
Incorrect Prediction:
Text: I caught this film late on a sat night/ Sunday morning with my brother.
We had been drinking. This is one of the best films for ripping apart I have
ever seen. From the 'luxury' ocean liner actually b...
True label: 1
Predicted label: tensor([0], device='cuda:0')
--------------------------------------------------
```