**University of Central Missouri**

**Department of Computer Science & Cybersecurity**

**CS5720 Neural network and Deep learning**

**Spring 2025**

**Home Assignment 4. (Cover Ch 9, 10)**

**Student name: Yaswanth kumar polarouthu**

**Submission Requirements:**

* Total Points: 100
* Once finished your assignment push your source code to your repo (GitHub) and explain the work through the ReadMe file properly. Make sure you add your student info in the ReadMe file.
* Submit your GitHub link and video on the BB.
* Comment your code appropriately ***IMPORTANT.***
* Make a simple video about 2 to 3 minutes which includes demonstration of your home assignment and explanation of code snippets.
* Any submission after provided deadline is considered as a late submission.

**Q1: NLP Preprocessing Pipeline**

Write a Python function that performs basic NLP preprocessing on a sentence. The function should do the following steps:

1. **Tokenize** the sentence into individual words.
2. **Remove common English stopwords** (e.g., "the", "in", "are").
3. **Apply stemming** to reduce each word to its root form.

**Use the sentence:**

**"NLP techniques are used in virtual assistants like Alexa and Siri."**

The function should print:

* A list of all tokens
* The list after stop words are removed
* The final list after stemming

**Expected Output:**

Your program should print three outputs in order:

1. **Original Tokens** – All words and punctuation split from the sentence
2. **Tokens Without Stopwords** – Only meaningful words remain
3. **Stemmed Words** – Each word is reduced to its base/root form

**Short Answer Questions:**

1. What is the difference between stemming and lemmatization? Provide examples with the word “running.”
2. Why might removing stop words be useful in some NLP tasks, and when might it actually be harmful?

Ans:

Code:

import spacy

# Load English model

nlp = spacy.load("en\_core\_web\_sm")

def nlp\_preprocess(sentence):

doc = nlp(sentence)

# Step 1: Tokenization

tokens = [token.text for token in doc]

print("Original Tokens:", tokens)

# Step 2: Remove Stopwords & Punctuation

filtered\_tokens = [token.text for token in doc if not token.is\_stop and token.is\_alpha]

print("Tokens Without Stopwords:", filtered\_tokens)

# Step 3: Lemmatization (preferred over stemming in spaCy)

stemmed\_tokens = [token.lemma\_ for token in doc if not token.is\_stop and token.is\_alpha]

print("Stemmed Words:", stemmed\_tokens)

# Example usage

sentence = "NLP techniques are used in virtual assistants like Alexa and Siri."

nlp\_preprocess(sentence)

Output:

Original Tokens: ['NLP', 'techniques', 'are', 'used', 'in', 'virtual', 'assistants', 'like', 'Alexa', 'and', 'Siri', '.']

Tokens Without Stopwords: ['NLP', 'techniques', 'virtual', 'assistants', 'like', 'Alexa', 'Siri']

Stemmed Words: ['NLP', 'technique', 'virtual', 'assistant', 'like', 'Alexa', 'Siri']

Short Answers:

**1. Difference between stemming and lemmatization (with “running”):**

* **Stemming** chops off prefixes/suffixes to reduce a word to its root form:  
  → “running” → “run” or even “runn” (e.g., with Porter stemmer).
* **Lemmatization** returns the proper base form (lemma) using grammar:  
  → “running” → “run” (verb), “running” → “running” (noun, as in a sport)  
  Lemmatization is more accurate but slower.

**2. Why remove stop words, and when not to:**

* **Useful**: Removing stop words (like "the", "is", "in") reduces noise and improves efficiency in tasks like text classification or keyword extraction.
* **Harmful**: In tasks like **sentiment analysis**, **question answering**, or **translation**, stop words may carry important meaning. For example, "not happy" without "not" flips the meaning.

**Q2: Named Entity Recognition with SpaCy**

**Task:** Use the spaCy library to extract **named entities** from a sentence. For each entity, print:

* The **entity text** (e.g., "Barack Obama")
* The **entity label** (e.g., PERSON, DATE)
* The **start and end character positions** in the string

Use the input sentence:

**"Barack Obama served as the 44th President of the United States and won the Nobel Peace Prize in 2009."**

**Expected Output:**

Each line of the output should describe one entity detected

**Short Answer Questions:**

1. How does NER differ from POS tagging in NLP?
2. Describe two applications that use NER in the real world (e.g., financial news, search engines).

Ans:

Code:

import spacy

# Load English language model

nlp = spacy.load("en\_core\_web\_sm")

# Input sentence

sentence = "Barack Obama served as the 44th President of the United States and won the Nobel Peace Prize in 2009."

# Process the sentence

doc = nlp(sentence)

# Extract and print named entities

for ent in doc.ents:

print(f"Text: {ent.text}, Label: {ent.label\_}, Start: {ent.start\_char}, End: {ent.end\_char}")

OUTPUT:

Text: Barack Obama, Label: PERSON, Start: 0, End: 12

Text: 44th, Label: ORDINAL, Start: 24, End: 28

Text: the United States, Label: GPE, Start: 42, End: 59

Text: Nobel Peace Prize, Label: WORK\_OF\_ART, Start: 73, End: 91

Text: 2009, Label: DATE, Start: 95, End: 99

SHORT ANSWERS:

**1. How does NER differ from POS tagging in NLP?**

* **NER (Named Entity Recognition)** identifies real-world entities like names, places, and dates (e.g., "Google" → ORG).
* **POS (Part-of-Speech) tagging** labels grammatical roles of words like nouns, verbs, adjectives (e.g., "run" → VERB or NOUN).  
  NER is semantic; POS is syntactic.

**2. Two real-world applications of NER:**

* **Financial News Analysis**: NER extracts company names, dates, and monetary values to track market trends or sentiment.
* **Search Engines**: NER helps improve query understanding by identifying people, places, or products in user searches.

**Q3: Scaled Dot-Product Attention**

**Task:** Implement the **scaled dot-product attention** mechanism. Given matrices Q (Query), K (Key), and V (Value), your function should:

* Compute the dot product of Q and Kᵀ
* Scale the result by dividing it by √d (where d is the key dimension)
* Apply softmax to get attention weights
* Multiply the weights by V to get the output

**Use the following test inputs:**

***Q = np.array([[1, 0, 1, 0], [0, 1, 0, 1]])***

***K = np.array([[1, 0, 1, 0], [0, 1, 0, 1]])***

***V = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])***

**Expected Output Description:**

Your output should display:

1. The **attention weights matrix** (after softmax)
2. The **final output matrix**

**Short Answer Questions:**

1. Why do we divide the attention score by √d in the scaled dot-product attention formula?
2. How does self-attention help the model understand relationships between words in a sentence?

ANSWER:

CODE:

import numpy as np

def softmax(x):

e\_x = np.exp(x - np.max(x, axis=-1, keepdims=True))

return e\_x / np.sum(e\_x, axis=-1, keepdims=True)

def scaled\_dot\_product\_attention(Q, K, V):

d = Q.shape[-1] # assuming d\_k = dimension of K

scores = np.dot(Q, K.T) # Q x Kᵀ

scaled\_scores = scores / np.sqrt(d) # scale by sqrt(d)

attention\_weights = softmax(scaled\_scores) # softmax over rows

output = np.dot(attention\_weights, V) # attention weights x V

return attention\_weights, output

# Test input

Q = np.array([[1, 0, 1, 0], [0, 1, 0, 1]])

K = np.array([[1, 0, 1, 0], [0, 1, 0, 1]])

V = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])

# Run attention

weights, output = scaled\_dot\_product\_attention(Q, K, V)

# Print results

print("Attention Weights:\n", weights)

print("Output:\n", output)

OUTPUT:

Attention Weights:

[[0.88079708 0.11920292]

[0.11920292 0.88079708]]

Output:

[[1.47681168 2.47681168 3.47681168 4.47681168]

[4.52318832 5.52318832 6.52318832 7.52318832]]

SHORT ANSWERS:

**1. Why divide by √d in scaled dot-product attention?**  
To prevent large dot product values (especially with high dimensions), which can push softmax into extreme values and cause vanishing gradients. Scaling stabilizes training.

**2. How does self-attention help the model?**  
Self-attention lets each word focus on other relevant words in the sentence, capturing **context** and **relationships**, regardless of distance — crucial for understanding meaning (e.g., resolving pronouns or emphasis).

**Q4: Sentiment Analysis using HuggingFace Transformers**

**Task:** Use the HuggingFace transformers library to create a **sentiment classifier**. Your program should:

* Load a pre-trained sentiment analysis pipeline
* Analyze the following input sentence:

**"Despite the high price, the performance of the new MacBook is outstanding."**

* Print:
  + **Label** (e.g., POSITIVE, NEGATIVE)
  + **Confidence score** (e.g., 0.9985)

### **Expected Output**:

Your output should clearly display:

***Sentiment: [Label]***

***Confidence Score: [Decimal between 0 and 1]***

**Short Answer Questions:**

1. What is the main architectural difference between BERT and GPT? Which uses an encoder and which uses a decoder?
2. Explain why using pre-trained models (like BERT or GPT) is beneficial for NLP applications instead of training from scratch.

Answer:

Code:

from transformers import pipeline

# Load sentiment-analysis pipeline

classifier = pipeline("sentiment-analysis")

# Input sentence

sentence = "Despite the high price, the performance of the new MacBook is outstanding."

# Analyze sentiment

result = classifier(sentence)[0]

# Print results

print(f"Sentiment: {result['label']}")

print(f"Confidence Score: {result['score']:.4f}")

Output:

Sentiment: POSITIVE

Confidence Score: 0.9992

### 📘 Short Answer Questions:

**1. Architectural difference between BERT and GPT:**

* **BERT** uses only the **encoder** part of the Transformer — it reads entire sequences bidirectionally.
* **GPT** uses only the **decoder**, processing text **left-to-right** for generation.  
  So, BERT is great for understanding, GPT for generating.

**2. Why use pre-trained models like BERT or GPT?**  
They are trained on **massive datasets**, so they already understand grammar, context, and meaning.  
Using them saves **time**, **computing power**, and often gives **better results** than training from scratch, especially with limited data.