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: Nagabandi Shravya

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

CODE:

```
import nltk

from nltk.tokenize import word_tokenize

from nltk.corpus import stopwords

from nltk.stem import PorterStemmer



# Download necessary resources

nltk.download('punkt', force=True)

# Download 'punkt_tab' specifically. It's no longer automatically included with 'punkt'.

nltk.download('punkt_tab', force=True)
```

```
nltk.download('stopwords', force=True)


def nlp_preprocess(sentence):
    # 1. Tokenization
    tokens = word_tokenize(sentence)
    print("Original Tokens:", tokens)

    # 2. Remove Stopwords
    stop_words = set(stopwords.words('english'))
    tokens_no_stop = [word for word in tokens if word.lower() not in stop_words]
    print("Tokens Without Stopwords:", tokens_no_stop)

    # 3. Apply Stemming
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in tokens_no_stop]
    print("Stemmed Words:", stemmed_tokens)

# Test sentence
sentence = "NLP techniques are used in virtual assistants like Alexa and Siri."
nlp_preprocess(sentence)
```

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

OUTPUT:

```
Original Tokens: ['NLP', 'techniques', 'are', 'used', 'in', 'virtual', 'assistants', 'like', 'Alexa', 'and', 'Siri', '.']
Tokens Without Stopwords: ['NLP', 'techniques', 'used', 'virtual', 'assistants', 'like', 'Alexa', 'Siri', '.']
Stemmed Words: ['nlp', 'techniqu', 'use', 'virtual', 'assist', 'like', 'alexa', 'siri', '.']
```

Short Answer Questions:

1. **What is the difference between stemming and lemmatization? Provide examples with the word "running."**

- **ANSWER:**

  temming cuts off word suffixes to reduce words to their root form, often without considering grammar.
    - Example: "running" → "run" using Porter Stemmer, or even "runn" with less accurate stemmers.
- Lemmatization uses vocabulary and morphological analysis to return the correct base or dictionary form (lemma) of a word.
    - Example: "running" → "run" (accurate and grammatically valid).

☑ Key difference: Lemmatization is more accurate but slower; stemming is faster but more aggressive and less precise.

2. **Why might removing stop words be useful in some NLP tasks, and when might it actually be harmful?**

- **ANSWER:**

  Useful:

  Removing stop words is helpful in tasks like document classification, sentiment analysis, and keyword extraction, where high-frequency, low-meaning words (e.g., "the", "is", "and") can add noise and reduce efficiency.
- Harmful:

  In tasks like machine translation, text summarization, or question answering, stop words can carry important grammatical or contextual meaning. Removing them may lead to a loss of critical information.

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."

CODE:

```
import spacy

# Load SpaCy's English NER 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 named entities
for ent in doc.ents:
    print(f"Text: {ent.text}, Label: {ent.label_}, Start: {ent.start_char}, End: {ent.end_char}")
```

Expected Output:

Each line of the output should describe one entity detected

OUTPUT:

```
Text: Barack Obama, Label: PERSON, Start: 0, End: 12
Text: 44th, Label: ORDINAL, Start: 27, End: 31
Text: the United States, Label: GPE, Start: 45, End: 62
Text: the Nobel Peace Prize, Label: WORK_OF_ART, Start: 71, End: 92
Text: 2009, Label: DATE, Start: 96, End: 100
```

Short Answer Questions:

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

Named Entity Recognition (NER) identifies **specific entities** like names of people, organizations, places, dates, etc., within text.

1. Example: "Barack Obama" → `PERSON`

Part-of-Speech (POS) tagging labels each word with its **grammatical role** (e.g., noun, verb, adjective).

2. Example: "Obama" → `NNP` (proper noun)

✅ In short:
NER → *What is this?*
POS → *What is its grammatical role?*

2.Describe two applications that use NER in the real world (e.g., financial news, search engines).

ANSWER:

Financial News Analysis:

NER helps extract company names, stock tickers, and dates from news to identify trends or events affecting markets.

Search Engines:

NER improves relevance by understanding that "Apple" can be a **company** (ORG) or a **fruit** (not an entity), helping disambiguate search results.

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 K^T
- Scale the result by dividing it by $\sqrt{d}$ (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]])

CODE:

```
import numpy as np

def softmax(x):
    e_x = np.exp(x - np.max(x, axis=-1, keepdims=True))
    return e_x / e_x.sum(axis=-1, keepdims=True)

def scaled_dot_product_attention(Q, K, V):
    d_k = Q.shape[-1]  # key dimension
    scores = np.dot(Q, K.T)  # 1. Dot product QK^T
    scaled_scores = scores / np.sqrt(d_k)  # 2. Scale by sqrt(d)
    attention_weights = softmax(scaled_scores)  # 3. Apply softmax
    output = np.dot(attention_weights, V)  # 4. Multiply by V
    return attention_weights, output

# 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]])

# Run attention
attention_weights, output = scaled_dot_product_attention(Q, K, V)

# Display results
print("Attention Weights:\n", attention_weights)
print("\nOutput Matrix:\n", output)
```

Expected Output Description:

Your output should display:

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

OUTPUT:

```
Attention Weights:
 [[0.73105858 0.26894142]
 [0.26894142 0.73105858]]

Output Matrix:
 [[2.07576569 3.07576569 4.07576569 5.07576569]
 [3.92423431 4.92423431 5.92423431 6.92423431]]
```

Short Answer Questions:

1. **Why do we divide the attention score by √d in the scaled dot-product attention formula?**

ANSWER:

- When the dimensionality (d) is large, the dot products can grow too large in magnitude, pushing softmax into regions with very small gradients.
- Scaling by √d stabilizes the softmax function, keeping values in a manageable range and improving training efficiency.

2. **How does self-attention help the model understand relationships between words in a sentence?**

ANSWER:

- Self-attention allows each word to focus on other relevant words in the sentence by computing weighted combinations of all word embeddings.
- For example, in the sentence *"The cat sat on the mat"*, the word *"sat"* can attend to *"cat"* to better understand the subject performing the action.

☑ This mechanism helps models capture contextual meaning, dependencies, and even long-distance relationships between words.

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)

CODE:

```
import warnings
from transformers import pipeline

# Suppress warnings for cleaner output
warnings.filterwarnings("ignore")

# Load a specific pre-trained sentiment model
model_name = "distilbert/distilbert-base-uncased-finetuned-sst-2-english"
classifier = pipeline("sentiment-analysis", model=model_name)

# Input text
text = "Despite the high price, the performance of the new MacBook is
outstanding."

# Run sentiment analysis
result = classifier(text)[0]

# Display result
print("Sentiment:", result['label'])
print("Confidence Score:", round(result['score'], 4))
```

Expected Output:

Your output should clearly display:

Sentiment: [Label]
Confidence Score: [Decimal between 0 and 1]
OUTPUT:

```
Device set to use cpu
Sentiment: POSITIVE
Confidence Score: 0.9998
```

Short Answer Questions:

1. **What is the main architectural difference between BERT and GPT? Which uses an encoder and which uses a decoder?**

ANSWER:

- BERT is based on the encoder part of the Transformer architecture.
  - It processes input bidirectionally (looking at the full sentence context).
- GPT is based on the decoder side.
  - It processes text in a unidirectional (left-to-right) way, good for text generation.

☑ In short:

- BERT → Encoder-based
- GPT → Decoder-based

2. **Explain why using pre-trained models (like BERT or GPT) is beneficial for NLP applications instead of training from scratch.**

ANSWER:

- Pre-trained models:
  - Are trained on large datasets (millions of texts)
  - Learn universal language representations
- Benefits:
  - Save time and resources
  - Require much less data for fine-tuning

- Perform well on a wide range of tasks like classification, Q&A, summarization

☑ Bottom line: You get state-of-the-art performance with minimal effort.