University of Central Missouri
Department of Computer Science & Cybersecurity

CS5720 Neural Networks and Deep Learning
Summer 2025

Home Assignment 3. (Cover Ch 7, 9)

Student name:M. Siddartha Reddy
700774070

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 BrightSpace.
- 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: Implementing an RNN for Text Generation

Task: Recurrent Neural Networks (RNNs) can generate sequences of text. You will train an **LSTM-based RNN** to predict the next character in a given text dataset.

1. Load a **text dataset** (e.g., "Shakespeare Sonnets", "The Little Prince").
2. Convert text into a **sequence of characters** (one-hot encoding or embeddings).
3. Define an **RNN model** using LSTM layers to predict the next character.
4. Train the model and generate new text by **sampling characters** one at a time.
5. Explain the role of **temperature scaling** in text generation and its effect on randomness.

Hint: *Use tensorflow.keras.layers.LSTM() for sequence modeling.*

Q2: 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."**

Stemming:
Chops suffixes using rules, may not produce real words.
Example: "running" → "run" (with PorterStemmer), "universal" → "univers" ❌

Lemmatization:
Uses vocabulary and grammar to return the dictionary base form.
Example: "running" → "run" (with WordNetLemmatizer)

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

Useful when:
You want to reduce noise in tasks like classification, topic modeling, or search.

Harmful when:
Words like "not", "is", "in" are important — e.g., in **sentiment analysis**, **translation**, or **named entity recognition**.

Q3: 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?**

NER (Named Entity Recognition) identifies and classifies named entities (e.g., people, organizations, locations) in text.

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

NER focuses on meaning, while **POS focuses on grammar**.

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

1. **Financial News Analysis**
   → Extracts company names, stock symbols, or monetary values for market trend analysis.

2. **Search Engines**
   → Recognizes named entities in queries (e.g., "restaurants in Paris") to improve relevant search results.

Q4: 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]])

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?**

- **Context:** In scaled dot-product attention, the attention scores are computed as the dot product of query (Q) and key (K) vectors.
- **Problem:** When the dimension d of these vectors is large, the dot products tend to have large variance and values grow in magnitude.
- **Effect:** This can push the softmax function into regions with very small gradients (because softmax saturates), making training harder.
- **Solution:** Dividing by sqrt{d} scales down the dot product values, keeping them in a range that softmax can handle better, which stabilizes gradients and improves training.

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

- **Self-attention** allows the model to **weigh the importance of each word relative to every other word** in the same sentence.
- For each word, it computes attention scores with all other words, effectively capturing **contextual dependencies**, regardless of distance.
- This helps the model learn which words influence the meaning of a given word — for example, subject-verb agreement or resolving pronouns.
- As a result, the model builds **rich contextualized representations** of words that reflect their relationships and meaning in the whole sentence.

Q5: 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?**

1. Main architectural difference between BERT and GPT:

- **BERT** uses a **Transformer encoder** architecture.
- **GPT** uses a **Transformer decoder** architecture.

2. Which uses encoder and which uses decoder?

- **BERT:** Encoder only.
- **GPT:** Decoder only.

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

Pre-trained models already learn rich language representations from large text corpora.

This saves time and computational resources.

They improve performance on downstream tasks, especially with limited labeled data.

They provide a strong starting point, requiring only fine-tuning for specific tasks.