**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:Jagatti Pawan Kalyan**

**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.”

Ans . **Stemming** cuts words to their root form, often creating incomplete words (e.g., "running" → "run" or "runn"), while **lemmatization** returns the dictionary base form, considering context (e.g., "running" → "run"). Removing **stop words** is useful in tasks like text classification or search engines, improving focus on meaningful terms. However, it can harm tasks like sentiment analysis or translation, where words like "not" are essential for meaning, potentially changing the context if removed improperly.

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

Ans. Removing stop words helps simplify text, reduce noise, and improve performance in tasks like text classification and topic modeling. However, it can be harmful in sentiment analysis or machine translation, where stop words like “not” or “no” affect meaning. Their removal may change the sentence’s intent or lead to errors.

**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?

Ans. **Named Entity Recognition (NER)** identifies and classifies entities like persons, organizations, and locations in text. For example, in "Apple released a new iPhone," NER tags "Apple" as an Organization.

**Part-of-Speech (POS) Tagging** labels each word with its grammatical role, such as noun, verb, or adjective. In the same sentence, "Apple" is tagged as a noun.

In short: **NER focuses on real-world entities**, while **POS tagging focuses on grammatical function**.

1. Describe two applications that use NER in the real world (e.g., financial news, search engines).
2. Ans. **Financial News Analysis**:  
   NER extracts company names, stock symbols, locations, and monetary values from news articles to help traders or analysts track market-relevant information automatically.
3. **Search Engines**:  
   NER identifies entities like people, brands, or places in user queries (e.g., "Hotels in Paris") to improve search relevance by understanding what the user is specifically referring to.

**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 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?

Ans. We divide the attention score by **√d (square root of the key dimension)** to prevent the dot products from becoming too large when the dimensionality (**d**) is high.

Without scaling, large dot products can push the softmax function into regions with very small gradients, causing **vanishing gradients** and slowing training. Scaling ensures more stable and effective gradient flow, allowing the model to learn better attention distributions.

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

Ans. Self-attention allows the model to weigh the importance of each word relative to every other word in a sentence. By doing this, it captures **contextual relationships**—such as dependencies, word order, and meaning—regardless of their position. This helps the model understand how words influence each other, enabling better comprehension of phrases, long-range dependencies, and nuanced meanings in the 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?

Ans. The main architectural difference is:

* **BERT** uses only the **Transformer encoder** stack and is designed for **bidirectional** context understanding (looking at words before and after).
* **GPT** uses only the **Transformer decoder** stack and is designed for **unidirectional** (left-to-right) text generation.

So, **BERT = encoder-only**, and **GPT = decoder-only** architecture.

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

Ans. Using pre-trained models like BERT or GPT is beneficial because they have already learned rich language patterns from massive datasets, saving time and computational resources. They provide strong general language understanding, enabling better performance on various NLP tasks with less labeled data through fine-tuning. Training from scratch requires huge data and compute power, while pre-trained models offer faster, more accurate, and cost-effective solutions for real-world applications.