**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:**

**NITHIN AARON KOMMIREDDY 700752900**

**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?

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

* **Stemming** chops off word endings to get to the root form, often resulting in non-words.
  + Example: "running" → "run" (using Porter Stemmer) or "runn" in some stemmers.
* **Lemmatization** uses vocabulary and grammar rules to return the base or dictionary form.
  + Example: "running" → "run" (as the lemma of a verb)

**Summary**:

* Stemming is faster but less accurate.
* Lemmatization is more accurate but slower (needs context and part of speech).

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

* **Useful**:
  + In tasks like **text classification**, **information retrieval**, or **topic modeling**, stop words (e.g., *the, is, in*) add noise and don’t contribute much meaning.
* **Harmful**:
  + In **sentiment analysis**, **question answering**, or **machine translation**, stop words can carry **important context or structure**, so removing them might hurt performance.

**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).

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

* **Named Entity Recognition (NER)** identifies **specific entities** in text like names of people, places, organizations, dates, etc.
  + Example: "Barack Obama" → **PERSON**
* **Part-of-Speech (POS) tagging** labels each word with its **grammatical role**, such as noun, verb, adjective, etc.
  + Example: "Obama" → **NNP** (Proper Noun, Singular)

**Summary**:

* **NER** finds *what* something is (a real-world entity).
* **POS tagging** shows *how* the word functions in a sentence.

**2. Describe two applications that use NER in the real world.**

**1. Financial News Analysis**  
NER helps identify company names, stock symbols, key people, and dates to extract market-moving insights from news articles.

**2. Search Engines**  
Search engines use NER to understand queries like “weather in Paris” — identifying **“Paris”** as a location — to return accurate 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 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?

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

We divide by **√d** (where *d* is the dimension of the key vectors) to **prevent the dot product values from becoming too large**.

* When *d* is large, the dot product of query and key vectors can have high variance.
* Large values in softmax lead to very **small gradients** due to saturation, making learning slow or unstable.

**Scaling by √d keeps the softmax more stable**, ensuring better gradient flow during training.

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

**Self-attention** lets the model **compare each word to every other word** in the sentence and assign weights based on relevance.

* For example, in the sentence: *"The cat sat on the mat."*  
  Self-attention helps the model realize that **"cat"** is more related to **"sat"** than to **"mat"**.

This allows the model to:

* Capture **contextual meaning**
* Handle **long-range dependencies**
* Understand **which words to focus on**, regardless of their position

**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]***

**OUTPUT:**

**No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (**[**https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-sst-2-english**](https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-sst-2-english)**).**

**Using a pipeline without specifying a model name and revision in production is not recommended.**

**/usr/local/lib/python3.11/dist-packages/huggingface\_hub/utils/\_auth.py:94: UserWarning:**

**The secret `HF\_TOKEN` does not exist in your Colab secrets.**

**To authenticate with the Hugging Face Hub, create a token in your settings tab (**[**https://huggingface.co/settings/tokens**](https://huggingface.co/settings/tokens)**), set it as secret in your Google Colab and restart your session.**

**You will be able to reuse this secret in all of your notebooks.**

**Please note that authentication is recommended but still optional to access public models or datasets.**

**warnings.warn(**

**config.json: 100%**

**629/629 [00:00<00:00, 47.5kB/s]**

**Xet Storage is enabled for this repo, but the 'hf\_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the package with: `pip install huggingface\_hub[hf\_xet]` or `pip install hf\_xet`**

**WARNING:huggingface\_hub.file\_download:Xet Storage is enabled for this repo, but the 'hf\_xet' package is not installed. Falling back to regular HTTP download. For better performance, install the package with: `pip install huggingface\_hub[hf\_xet]` or `pip install hf\_xet`**

**model.safetensors: 100%**

**268M/268M [00:01<00:00, 235MB/s]**

**tokenizer\_config.json: 100%**

**48.0/48.0 [00:00<00:00, 4.11kB/s]**

**vocab.txt: 100%**

**232k/232k [00:00<00:00, 14.6MB/s]**

**Device set to use cuda:0**

**Sentiment: POSITIVE**

**Confidence Score: 0.9998302459716797**

**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.

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

* **BERT** (Bidirectional Encoder Representations from Transformers):
  + **Architecture**: BERT uses only the **encoder** part of the Transformer architecture.
  + **Main Feature**: BERT processes text **bidirectionally**, meaning it considers the entire context of a word (both left and right) at once. This makes it effective for understanding context and relationships between words in a sentence.
* **GPT** (Generative Pretrained Transformer):
  + **Architecture**: GPT uses only the **decoder** part of the Transformer architecture.
  + **Main Feature**: GPT generates text in a **left-to-right** manner, meaning it predicts the next word based on previous words. This makes GPT great for tasks like text generation.

**Summary**:

* **BERT** = **Encoder** + **Bidirectional** processing
* **GPT** = **Decoder** + **Autoregressive** (left-to-right) generation

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

* **Pre-trained models** are **already trained on massive amounts of data**, capturing complex language patterns, syntactic rules, and semantic meaning. This brings several benefits:
  1. **Time and Cost Efficiency**: Training from scratch requires vast computational resources and time. Pre-trained models reduce this cost significantly.
  2. **Better Performance**: Pre-trained models are often much more powerful than models trained from scratch because they have already learned general language features and can be fine-tuned for specific tasks.
  3. **Transfer Learning**: By fine-tuning a pre-trained model on a specific task (e.g., sentiment analysis), you leverage knowledge gained from general language understanding to perform well on task-specific data, improving performance and reducing the amount of data needed.
  4. **Generalization**: Pre-trained models generalize better to unseen data, as they have learned broader patterns in language rather than memorizing specifics.