Q1: Text Generation using LSTM-based RNN

```
import tensorflow as tf
import numpy as np

# Load a smaller portion of Shakespeare
text = open(tf.keras.utils.get_file("shakespeare.txt",
       "https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt"),
       'rb').read().decode('utf-8')[:100000]  # take only first 100,000 chars

vocab = sorted(set(text))
char2idx = {u:i for i,u in enumerate(vocab)}
idx2char = np.array(vocab)
text_as_int = np.array([char2idx[c] for c in text])

seq_length = 50
sequences = tf.data.Dataset.from_tensor_slices(text_as_int).batch(seq_length+1, drop_remainder=True)
dataset = sequences.map(lambda x: (x[:-1], x[1:])).shuffle(10000).batch(32, drop_remainder=True)
dataset = dataset.take(100)  # take only 100 batches for speed

model = tf.keras.Sequential([
    tf.keras.layers.Embedding(len(vocab), 128),
    tf.keras.layers.LSTM(256, return_sequences=True, stateful=True, recurrent_initializer='glorot_uniform'),
    tf.keras.layers.Dense(len(vocab))
])

model.compile(optimizer='adam', loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True))
model.fit(dataset, epochs=1)

# model.build(tf.TensorShape([1, None])) # Removed this line
model.layers[1].reset_states() # Call reset_states on the LSTM layer

def generate_text(model, start="ROMEO:", temp=1.0):
    input_eval = tf.expand_dims(tf.constant([char2idx[c] for c in start]), 0)
    text_out = []
    model.layers[1].reset_states() # Call reset_states on the LSTM layer
    for _ in range(300):  # shorter generation
        preds = model(input_eval)[:, -1, :] / temp
        next_id = tf.random.categorical(preds, 1)[-1,0].numpy()
        input_eval = tf.expand_dims([next_id], 0)
        text_out.append(idx2char[next_id])
    return start + ''.join(text_out)

print(generate_text(model))
```

```
61/61 ━━━━━━━━━━━━━━━━━━━━ 22s 261ms/step - loss: 3.5098
ROMEO:G,JmQP?&KDPvY;-UDGT;zPqQ'xqK,NLuHDxkMzDjJY&C,QPDjrjO;xOqvIVyV'jPMhHJMvFnPRDDSvKAYxxGDCFzzjKPO&PQjQ&P,kjP!wqYQQNPq;gjBxHVqvVPQ
hQKx&v&GWFQQdQ&rKQRHQj&PW?TPN?-&xPCMCcjROJ?Vpx;T
&JTFpD.pPGkEIVmMD&zYQQxQN;w&BGD?JQpVPQTzQVkIlCGAQFqQ!DLKM?vPFKzjTvVf&QKQzPLPgjY&LYQYqjkGqGCKjQlL;QPqFHK&vE
```

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

Stemming is a crude process that chops off word endings to reduce words to their root form, often resulting in non-words.

Example: “running” → run (or sometimes runn depending on the stemmer)

Lemmatization is more sophisticated; it reduces a word to its dictionary (lemma) form using vocabulary and grammar rules.

Example: “running” → run (with part-of-speech tagging to determine the correct form)

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

Useful: Removing stop words (e.g., “the,” “is,” “and”) helps reduce noise and dimensionality in tasks like document classification or topic modeling, where such words add little meaning.

Harmful: In tasks like sentiment analysis, machine translation, or question answering, stop words can carry crucial meaning (e.g., “not” in “not good”), so removing them may distort the text's intent.

Q2: NLP Preprocessing Pipeline

```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer
```

```
# Explicitly add the default NLTK data path
nltk.data.path.append('/root/nltk_data')

nltk.download('punkt')
nltk.download('stopwords')

sentence = "NLP techniques are used in virtual assistants like Alexa and Siri."
tokens = word_tokenize(sentence)
print("Original Tokens:", tokens)
filtered = [w for w in tokens if w.lower() not in stopwords.words('english')]
print("Tokens without Stopwords:", filtered)
stemmer = PorterStemmer()
stemmed = [stemmer.stem(word) for word in filtered]
print("Stemmed Words:", stemmed)
```

```
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', '.']
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Q3: Named Entity Recognition with spaCy

```
import spacy
nlp = spacy.load("en_core_web_sm")
text = "Barack Obama served as the 44th President of the United States and won the Nobel Peace Prize in 2009."
doc = nlp(text)
for ent in doc.ents:
    print(f"Entity: {ent.text}, Label: {ent.label_}, Start: {ent.start_char}, End: {ent.end_char}")
```

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

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

NER (Named Entity Recognition) identifies and classifies named entities in text into categories like person names, organizations, locations, dates, etc.

Example: “Apple Inc. was founded in California.” →

"Apple Inc." → Organization, "California" → Location

POS (Part-of-Speech) tagging assigns grammatical roles to each word in a sentence, such as noun, verb, adjective, etc.

Example: “Apple Inc. was founded in California.” →

"Apple" → NNP (proper noun), "was" → VBD (verb, past tense)

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

Financial News Analysis: NER helps identify companies, stock symbols, and locations in articles to automate market sentiment analysis or track events affecting specific entities.

Search Engines: NER improves search relevance by recognizing and prioritizing named entities in queries (e.g., distinguishing "Amazon" the company vs. the rainforest).

Q4: Scaled Dot-Product Attention

```
import numpy as np
def scaled_dot_product_attention(Q, K, V):
    d_k = Q.shape[-1]
    scores = np.dot(Q, K.T) / np.sqrt(d_k)
    weights = np.exp(scores) / np.sum(np.exp(scores), axis=1, keepdims=True)
    output = np.dot(weights, V)
    return weights, output

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]])
weights, output = scaled_dot_product_attention(Q, K, V)
print("Attention Weights:\n", weights)
print("Output:\n", output)
```

```
Attention Weights:
 [[0.73105858 0.26894142]
 [0.26894142 0.73105858]]
Output:
 [[2.07576569 3.07576569 4.07576569 5.07576569]
 [3.92423431 4.92423431 5.92423431 6.92423431]]
```

1. Why do we divide the attention score by √d in the scaled dot-product attention formula? Answer: • Prevents very large dot products which can make softmax outputs too sharp. • Without scaling, gradients may vanish or explode, making training unstable.
2. How does self-attention help the model understand relationships between words in a sentence? Answer: • Self-attention lets each word "attend" to every other word in the sentence. • Captures contextual meaning: o In "The bank of the river", "bank" gets different meaning than in "money bank".

Q5: Sentiment Analysis using HuggingFace Transformers

```
from transformers import pipeline
classifier = pipeline('sentiment-analysis')
result = classifier("Despite the high price, the performance of the new MacBook is outstanding.")[0]
print(f"Sentiment: {result['label']}, Confidence Score: {result['score']:.4f}")
```

```
No model was supplied, defaulted to distilbert/distilbert-base-uncased-finetuned-sst-2-english and revision 714eb0f (https://huggin
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), set it as
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, 36.1kB/s]
model.safetensors: 100%    268M/268M [00:06<00:00, 48.0MB/s]
tokenizer_config.json: 100%    48.0/48.0 [00:00<00:00, 591B/s]
vocab.txt: 100%    232k/232k [00:00<00:00, 2.80MB/s]
Device set to use cpu
Sentiment: POSITIVE, Confidence Score: 0.9998
```

1. What is the main architectural difference between BERT and GPT? Which uses an encoder and which uses a decoder? Answer: Feature BERT GPT Core Design Encoder-only (bidirectional) Decoder-only (autoregressive) Pre-training Masked Language Modeling Next Word Prediction (causal) Use-case Understanding context (e.g., QA, classification) Text generation, completion
2. Explain why using pre-trained models (like BERT or GPT) is beneficial for NLP applications instead of training from scratch. Answer: • Massive training cost saved: Pre-trained on billions of words. • Quick fine-tuning: Just a few labeled examples needed for your task. • State-of-the-art performance: Works out of the box for most NLP tasks.