

```

# STEP 1: Load the text dataset
import tensorflow as tf
import numpy as np
import os

# Download and read the text data
path = tf.keras.utils.get_file("little_prince.txt", "https://www.gutenberg.org/files/1417/1417-0.txt")
text = open(path, encoding='utf-8').read().lower()
text = text[:100000] # ▼ Use only the first 100,000 characters for faster training
print(f'Reduced text length: {len(text)} characters')

# STEP 2: Preprocess the data
vocab = sorted(set(text))
char2idx = {char: idx for idx, char in enumerate(vocab)}
idx2char = np.array(vocab)
text_as_int = np.array([char2idx[c] for c in text])

# Create training examples and targets
seq_length = 100
char_dataset = tf.data.Dataset.from_tensor_slices(text_as_int)
sequences = char_dataset.batch(seq_length+1, drop_remainder=True)

def split_input_target(chunk):
    return chunk[:-1], chunk[1:]

dataset = sequences.map(split_input_target)

# STEP 3: Build the LSTM model (optimized)
BATCH_SIZE = 32 # ▼ Reduced
BUFFER_SIZE = 10000
EMBEDDING_DIM = 128 # ▼ Reduced
RNN_UNITS = 256 # ▼ Reduced
VOCAB_SIZE = len(vocab)

dataset = dataset.shuffle(BUFFER_SIZE).batch(BATCH_SIZE, drop_remainder=True)

def build_model(vocab_size, embedding_dim, rnn_units, batch_size):
    return tf.keras.Sequential([
        tf.keras.layers.Input(shape=(None,), batch_size=batch_size),
        tf.keras.layers.Embedding(vocab_size, embedding_dim),
        tf.keras.layers.LSTM(rnn_units, return_sequences=True, stateful=True, recurrent_initializer='glorot_uniform'),
        tf.keras.layers.Dense(vocab_size)
    ])

model = build_model(VOCAB_SIZE, EMBEDDING_DIM, RNN_UNITS, BATCH_SIZE)

# STEP 4: Compile and train the model
def loss(labels, logits):
    return tf.keras.losses.sparse_categorical_crossentropy(labels, logits, from_logits=True)

model.compile(optimizer='adam', loss=loss)

# Checkpoint setup
checkpoint_dir = './training_checkpoints'
checkpoint_prefix = os.path.join(checkpoint_dir, "ckpt_{epoch}.weights.h5")

checkpoint_callback = tf.keras.callbacks.ModelCheckpoint(
    filepath=checkpoint_prefix,
    save_weights_only=True
)

EPOCHS = 1 # ▼ Reduced to 1 epoch for quick testing
history = model.fit(dataset, epochs=EPOCHS, callbacks=[checkpoint_callback])

# STEP 5: Text Generation with Temperature Scaling
def generate_text(model, start_string, temperature=1.0):
    model = build_model(VOCAB_SIZE, EMBEDDING_DIM, RNN_UNITS, batch_size=1)
    model.load_weights(os.path.join(checkpoint_dir, "ckpt_1.weights.h5"))
    model.build(tf.TensorShape([1, None]))

    input_eval = [char2idx[s] for s in start_string.lower()]
    input_eval = tf.expand_dims(input_eval, 0)

    text_generated = []

    for i in range(500):
        predictions = model(input_eval)
        predictions = tf.squeeze(predictions, 0)

        predictions = predictions / temperature
        predicted_id = tf.random.categorical(predictions, num_samples=1)[-1, 0].numpy()

        input_eval = tf.concat([input_eval, [[predicted_id]]], 0)
        text_generated.append(index2char[predicted_id])

    return start_string + ''.join(index2char[c] for c in text_generated)

```

```

input_eval = tf.expand_dims([predicted_id], 0)
text_generated.append(idx2char[predicted_id])

return start_string + ''.join(text_generated)

# Generate sample outputs
print("\n--- Generated Text with Temperature = 0.5 ---\n")
print(generate_text(model, start_string="Once upon a time ", temperature=0.5))

print("\n--- Generated Text with Temperature = 1.0 ---\n")
print(generate_text(model, start_string="Once upon a time ", temperature=1.0))

```

 Downloading data from <https://www.gutenberg.org/files/1417/1417-0.txt>
692241/692241 **0s** 1us/step
 Reduced text length: 100000 characters
30/30 **13s** 363ms/step - loss: 3.4814

--- Generated Text with Temperature = 0.5 ---

Once upon a time teere oe e t o et a dmet , o sae a aloaoheionose g ah a he on itas neta nnte a h ele s ne oist l i
 iheni nt ese sah e oai s n oo ieernto o ae e a ieot i
 eer tre ala ooa ade e ahnt hn te al soa feaedns ahe n thoei lo t hsai a hoa ar eoa eef g a ualo dan seh h e a

--- Generated Text with Temperature = 1.0 ---

Once upon a time h y l ienbs ihiago kaesmf,swghctedct.lnehelsh id winneosltrieatob"t fol sl luo oa amie-yeny ou.fr.p
 htn isolmeima nlhwos sl ylavasacwen ebcgehel iooidrcteen "a p y ahc 'n oh rm
 tynsiaiynta ftpedeater yawelhpeeo pgaeonre tbato afupine'anciar layvw ndaionrmhy sdh tet harlwx pueyy t t oehneu f
 ,kep. letssos ohuch,eeptnesle nhhnetsa,silcedhn.aseleares
 aeel b atta elrhinrleayat -foo w tn dohwsws aaa,gahe t.foe hs ar id oldea- emg tsnrkad aatiliorfwoecrg4haernd.ich i naot

```

import nltk
# Download all essential resources safely
nltk.download('punkt')
nltk.download('punkt_tab')
nltk.download('stopwords')
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer

# Download required NLTK data (only once)
nltk.download('punkt')
nltk.download('stopwords')


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

    # Step 2: Remove Stopwords
    stop_words = set(stopwords.words('english'))
    tokens_without_stopwords = [word for word in tokens if word.lower() not in stop_words]
    print("\n2. Tokens Without Stopwords:")
    print(tokens_without_stopwords)

    # Step 3: Stemming
    stemmer = PorterStemmer()
    stemmed_tokens = [stemmer.stem(word) for word in tokens_without_stopwords]
    print("\n3. Stemmed Words:")
    print(stemmed_tokens)

# Sample sentence
sentence = "NLP techniques are used in virtual assistants like Alexa and Siri."
preprocess_nlp(sentence)

```

 [nltk_data] Downloading package punkt to /root/nltk_data...
 [nltk_data] Package punkt is already up-to-date!
 [nltk_data] Downloading package punkt_tab to /root/nltk_data...
 [nltk_data] Unzipping tokenizers/punkt_tab.zip.
 1. Original Tokens:
 ['NLP', 'techniques', 'are', 'used', 'in', 'virtual', 'assistants', 'like', 'Alexa', 'and', 'Siri', '.']
 2. Tokens Without Stopwords:
 ['NLP', 'techniques', 'used', 'virtual', 'assistants', 'like', 'Alexa', 'Siri', '.']
 3. Stemmed Words:
 ['nlp', 'techniqu', 'use', 'virtual', 'assist', 'like', 'alexa', 'siri', '.']
 [nltk_data] Downloading package stopwords to /root/nltk_data...
 [nltk_data] Package stopwords is already up-to-date!
 [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!
```

```
import spacy
```

```
# Load English language 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."
```

```
# Apply NLP pipeline
doc = nlp(sentence)
```

```
# Extract and print entities
for ent in doc.ents:
    print(f"Text: {ent.text}, Label: {ent.label_}, Start: {ent.start_char}, End: {ent.end_char}")
```

```
→ 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
```

```
import numpy as np
```

```
def scaled_dot_product_attention(Q, K, V):
    # Step 1: Compute dot product of Q and KT
    matmul_qk = np.dot(Q, K.T)

    # Step 2: Scale by sqrt(d), where d is the key dimension
    d_k = K.shape[-1]
    scaled_scores = matmul_qk / np.sqrt(d_k)
```

```
    # Step 3: Apply softmax to get attention weights
    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)
```

```
    attention_weights = softmax(scaled_scores)
```

```
    # Step 4: Multiply attention weights by V
    output = np.dot(attention_weights, 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 the function
attention_weights, output = scaled_dot_product_attention(Q, K, V)
```

```
# Print results
print("1. Attention Weights:\n", attention_weights)
print("\n2. Final Output:\n", output)
```

```
→ 1. Attention Weights:
[[0.73105858 0.26894142]
 [0.26894142 0.73105858]]

2. Final Output:
[[2.07576569 3.07576569 4.07576569 5.07576569]
 [3.92423431 4.92423431 5.92423431 6.92423431]]
```

```
# Sentiment Analysis using HuggingFace Transformers
```

```
from transformers import pipeline
```

```
# Load a pre-trained sentiment analysis pipeline
sentiment_pipeline = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")
```

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

```
# Analyze sentiment
result = sentiment_pipeline(sentence)[0]
```

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

```
🔗 /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),
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, 15.0kB/s]
model.safetensors: 100%                         268M/268M [00:03<00:00, 110MB/s]
tokenizer_config.json: 100%                     48.0/48.0 [00:00<00:00, 1.02kB/s]
vocab.txt: 100%                                232k/232k [00:00<00:00, 2.45MB/s]
Device set to use cpu
Sentiment: POSITIVE
Confidence Score: 0.9998
```