LSTM Language Model Report

Task 1. Dataset Acquisition

For this task, the ***Star Wars*** dataset from the Hugging Face Hub has been chosen, specifically the "myothiha/starwars" collection. This dataset is publicly available on the Hugging Face platform, which hosts a wide range of datasets. It contains 25,197 text samples, primarily consisting of dialogues and narratives from the Star Wars universe, making it ideal for language modeling tasks.

The dataset is divided into three parts:

- The **training set** contains 7,860 text samples.
- The **validation set** contains 8,101 samples.
- The **test set** contains 9,236 samples.

These text samples make the dataset well-suited for tasks such as text generation.

Dataset Source: Hugging Face Datasets, *myothiha/starwars*
Available at: https://huggingface.co/datasets/myothiha/starwars
This dataset is publicly available, with proper credit given to the creator, *myothiha*.

Task 2. Model Training

1) Preprocessing

To prepare the Star Wars dataset for training, the preprocessing steps were broken down into three main parts: **tokenization**, **numericalization**, and **batch preparation**.

- **Tokenization**: The raw text was tokenized using the ***basic_english*** tokenizer from the ***torchtext*** library. This tokenizer splits the text based on spaces and punctuation, breaking each sentence into individual tokens.For example, the sentence ***"Yes, I see,"*** is tokenized into ***['yes', ',', 'i', 'see', ',']***. The dataset was then mapped with a custom function to apply tokenization to each sentence, resulting in a tokenized dataset.

- **Numericalization**: After tokenization, the dataset was converted into numerical format, where each token was assigned an index from a vocabulary. The vocabulary was created using ***torchtext.vocab.build_vocab_from_iterator*** by analyzing the tokens in the training data. Only words that appeared at least three times were included in the vocabulary, ensuring that rare or irrelevant tokens were excluded. Special tokens, ***<unk>***

(unknown) and ***<eos> (end-of-sentence)*** were added to manage words not in the vocabulary and mark sentence boundaries. The vocabulary was then used to convert each token in the dataset into its corresponding index. For example, the word ***"yes"*** might be assigned the ***index 5***, and ***the comma*** might be assigned the ***index 7***.

- **Batch Preparation:** To prepare the data for the model, the tokenized and numericalized dataset was organized into batches of 32 samples, with each batch containing sequences of tokens. The batch preparation function ensured that each batch had an equal number of tokens by trimming extra tokens and reshaping the data into a tensor of shape ***[batch_size, seq_len]***, ready for training.

2) Model Architecture and Training Process

- **Model Architecture:** The model used for this language modeling task is ***LSTM-based language model***. It consists of the following components:
    - **Embedding Layer:** Converts the input token indices into dense vectors of fixed size (1024 in this case). This allows the model to capture semantic relationships between words.
    - **LSTM Layer**: The LSTM layer processes the embedded input with 1024 hidden units across 2 layers. It learns temporal dependencies between tokens by updating the hidden states at each time step, capturing the context and style of the text.
    - **Dropout Layer:** A dropout layer with a rate of 0.65 is applied after the embedding and LSTM layers to reduce overfitting.
    - **Fully Connected Layer:** After the LSTM processes the sequence, its output is passed through a fully connected (linear) layer to predict the next token in the sequence. The output size of this layer matches the vocabulary size, providing a probability distribution for the next possible word.
    - **Initialization**: Weights are initialized randomly within a specific range, while biases are initialized to zero to ensure stable training.

- **Training Process:** The training process uses a sequence of operations to train the model:
    - **Loss Function:** ***Cross-Entropy Loss*** is used, which is a standard loss function for classification tasks like language modeling. This loss function measures how far the model's predictions are from the actual target tokens.
    - **Optimizer:** The ***Adam optimizer*** is used to adjust the model's weights based on the computed gradients. Adam is chosen for its efficiency in handling large datasets and its ability to dynamically adjust the learning rate, which helps in faster and more stable training.
    - **Gradient Clipping:** To prevent exploding gradients, which could destabilize training, gradient clipping is applied. If the gradients exceed a threshold of 0.25, they are clipped, ensuring they remain within a manageable range.
    - **Training and Evaluation:** The model is trained for 30 epochs. In each epoch, the training loss is computed by processing the training data in batches, and the evaluation loss is calculated using the validation set. The learning rate is dynamically adjusted using the ***ReduceLROnPlateau*** scheduler. If the validation loss stops improving for a few rounds, it lowers the learning rate, helping the training go more smoothly.
    - **Perplexity:** The performance of the model was evaluated using perplexity, which is a measure of how well the model predicts the next token. Lower perplexity values indicate better model performance.
    - **Model Saving:** The model's parameters are saved whenever the validation loss improves, ensuring that the best-performing model is retained.

Task 3. Web Application Interface with the Language Model

Overview:

The web application allows users to input a text prompt, which the trained LSTM language model uses to generate a continuation. The application demonstrates how the model attempts to produce text that is contextually relevant and flows from the given prompt, though the quality of the output may vary.

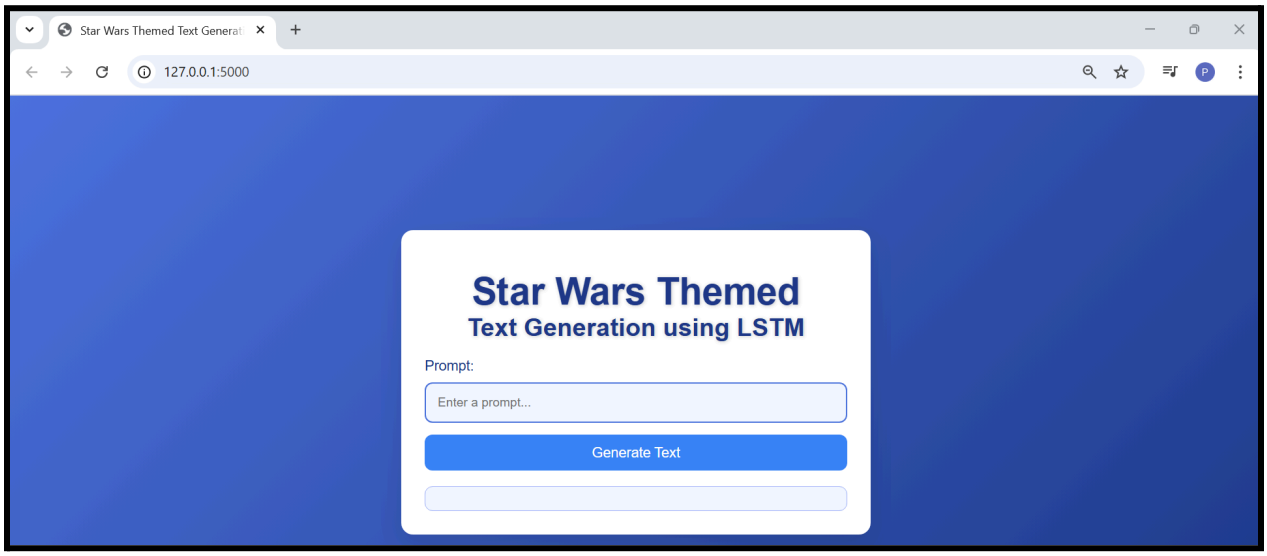
How it Works:

1. **User Input:**
    - The user sees a simple text box where they can type ***a prompt*** (e.g., "yoda is").
2. **Text Generation:**
    - When the user clicks ***'Generate Text'***, the backend processes the prompt.
    - The language model generates text by predicting the next word, continuously building the sequence until a certain condition is met (e.g., reaching a max length or encountering an end-of-sequence token).
3. **Output Display:**
    - The generated text is displayed immediately below the input box, allowing the user to read the continuation of their prompt.

Sample Screenshots:

Screenshot 1: Homepage

This screenshot shows the initial homepage of the web application, where the user can input a text prompt.



Screenshot 2: Text Generation Output

This screenshot displays the result after the user entered the prompt ***"yoda is"*** and clicked ***"Generate Text."*** The model processes the input and generates a continuation of the text as an output.