Asian Institute of Technology (AIT)

Artificial Intelligence: Natural Language Understanding (NLU)

Assignment 2 (A2): Language Model

Submitted To:

Dr. Chaklam Silpasuwanchai

Assistant Professor

Computer Science and Information Management

Asian Institute of Technology

chaklam@ait.asia

Submitted By:

Tisa Bajracharya

St126686

GitHub Link: https://github.com/bajratisa/NLP_assignment

1. Scope of Project

The objective of this project is to develop a natural language processing (NLP) system capable of generating coherent text in the style of J.K. Rowling's Harry Potter and the Sorcerer's Stone. The scope includes:

- **Data Preparation:** Acquiring and cleaning raw text data to build a usable vocabulary.
- **Model Development:** Designing and training a Long Short-Term Memory (LSTM) neural network to predict the next word in a sequence.
- **Web Application:** deploying the trained model via a user-friendly web interface that allows real-time interaction and text generation.
- **Experimentation:** Implementing temperature sampling to allow users to control the diversity and creativity of the generated output.

2. Training Data

The model was trained on the full text of the first book in the Harry Potter series.

- **Source:** Harry Potter and the Sorcerer's Stone (Text file).
- **Preprocessing:**
    - **Tokenization:** The text was split into individual tokens (words and punctuation) using regular expressions.
    - **Normalization:** All text was converted to lowercase to ensure consistency.
    - **Vocabulary Construction:** A dictionary was built containing all unique words that appeared at least **3 times** in the text.
    - **Special Tokens:**
        - <unk>: Used to represent rare or unknown words not in the vocabulary.
        - <eos>: Marks the end of a sentence.
- **Statistics:**
    - **Vocabulary Size:** 2,154 unique tokens.
    - **Training Batches:** The data was batched to allow for efficient parallel processing during training.

3. Workflow

The project followed a standard Machine Learning pipeline:

- **Data Ingestion:** Loading the raw .txt file.

- **Preprocessing:** Tokenizing text and converting words into numerical indices (Word IDs).
- **Model Architecture Design:** Defining the LSTM class with embedding and dropout layers.
- **Training:**
  - Optimizing weights using the Adam optimizer.
  - Minimizing Cross-Entropy Loss.
  - Monitoring Perplexity (PPL) on validation data to prevent overfitting.
- **Inference:** Generating text by feeding a "seed" prompt into the model and sampling the next word iteratively.
- **Deployment:** Integrating the model into a Dash web application for end-user testing.

4. Libraries and Tools

The following Python libraries were utilized to build the system:

- **PyTorch (torch):**
  - Purpose: The core deep learning framework used for defining the LSTM architecture, managing tensors, and performing backpropagation.
  - Reason for use: It offers dynamic computation graphs and efficient GPU acceleration (if available).
- **Dash (dash, dash_html_components, dash_core_components):**
  - Purpose: A framework for building the web-based user interface.
  - Reason for use: It allows for the rapid creation of interactive data apps directly in Python without needing extensive JavaScript knowledge.
- **Regular Expressions (re):**
  - Purpose: Used for text cleaning and tokenization.
  - Reason for use: Provides a fast and flexible way to split strings based on patterns (e.g., separating punctuation from words).
- **Pickle (pickle):**
  - Purpose: Object serialization.
  - Reason for use: Used to save and load the custom Vocabulary object so the web app understands the mapping between words and numbers.

5. Model Architecture

Model Used: Long Short-Term Memory (LSTM)

An LSTM is a specialized type of Recurrent Neural Network (RNN) capable of learning long-term dependencies.

Structure:

- **Embedding Layer:** Maps input word indices to dense vectors of size 1024.
- **LSTM Layers:** 2 stacked layers with 1024 hidden units each.
- **Dropout Layer:** Applied with a probability of 0.65 to reduce overfitting.
- **Fully Connected Layer:** Maps the hidden state to the output vocabulary size.

Advantages

- **Context Management:** Unlike standard RNNs, LSTMs have "gates" (input, output, forget) that allow them to remember important information over long sequences and forget irrelevant data.
- **Sequence Handling:** They are explicitly designed for sequential data like text, making them superior to traditional statistical models (like N-grams) for this task.

Disadvantages

- **Computational Cost:** LSTMs train slower than simpler models because of the complex gate operations inside each cell.
- **Vanishing Gradient:** While better than standard RNNs, LSTMs can still struggle with extremely long sequences compared to modern Transformer architectures (like GPT).

6. Conclusion

The project successfully delivered a functional language model and an interactive web application.

- **Performance:** The model achieved a Validation Perplexity of approximately 81.4, indicating it has learned the statistical structure of the training text effectively.
- **Application:** The web interface allows users to generate 10 distinct variations of text simultaneously, demonstrating the model's ability to balance coherent prediction (low temperature) with creative exploration (high temperature).

How to Run

To launch the application locally:

1. Ensure all dependencies are installed (pip install torch dash).
2. Navigate to the project directory.
3. Run the command: python app/app.py
4. Open a browser to http://127.0.0.1:8050/.

Core Function: generate_text

This function is responsible for the inference logic.

Parameters:

- prompt (str): The initial text string provided by the user.
- max_seq_len (int): The maximum number of words to generate (set to 25-50).
- temperature (float): Controls randomness.
  - **< 1.0**: Makes the model more confident/conservative.
  - **> 1.0**: Makes the model more random/creative.
- model: The trained PyTorch LSTM model.
- vocab: The dictionary mapping words to IDs.

Output:

- Returns a string containing the completed sentence. Special tokens (<unk> and <eos>) are filtered out for cleaner display.