Report: Implementing a Text Generator on the Persian Wikipedia Dataset

1. Introduction

- The objective of this project is to develop a text generator on the Persian Wikipedia dataset. The model will be implemented from scratch, and pre-trained networks will not be used. The report outlines the steps taken to create the dataset, design the model, train and evaluate it.

2. Dataset Creation:

- The goal of this step is to create a structured and efficient text generation dataset from the Persian Wikipedia. This dataset will be used to train the neural network model, enabling it to learn patterns and generate text in Persian.
- **Process:**

  a) **Vocabulary Class**:

  - Initialization:

    - **Special Tokens**: The vocabulary is initialized with special tokens (`<PAD>`, `<SOS>`, `<EOS>`, and `<UNK>`) which serve specific roles during training and inference. `<PAD>` is used for padding sequences to the same length, `<SOS>` marks the start of a sequence, `<EOS>` indicates the end of a sequence, and `<UNK>` represents unknown tokens.
    - **Frequency Threshold and Maximum Size**: These parameters are set to control which words are included in the vocabulary based on their frequency in the text. The `freq_threshold` ensures that only words appearing more than a certain number of times are included, while `max_size` limits the total number of words in the vocabulary.
    - **Tokenizer**: A tokenizer (such as `hazm.WordTokenizer()`) is used to split text into individual tokens (words).

  - Build **Vocabulary**:

    - **Token Frequency Calculation**: The text is tokenized, and the frequency of each token is calculated. Only tokens that exceed the frequency threshold are considered.
    - **Vocabulary Limitation**: The tokens are sorted by frequency, and the most frequent tokens are selected up to the maximum size limit. This ensures that the vocabulary is both comprehensive and manageable.
    - **Index Assignment**: Each token is assigned a unique index, starting from the index 4 (following the special tokens).

  - Get **Vocabularies**:

    - **Retrieve Mappings**: Two mappings are maintained: one from tokens to indices (`stoi`) and another from indices to tokens (`itos`). These mappings are essential for converting between tokenized text and numerical representations.

- Numericalize:
  - **Token to Index Conversion**: Given a text, the tokenizer splits it into tokens, which are then converted to their corresponding indices using the `stoi` mapping. If a token is not found in the vocabulary, it is mapped to the index for `<UNK>`.

b) **Custom Dataset Class**:

- Initialization:
  - **Input Text and Vocabulary**: The class is initialized with the input text and the vocabulary created earlier.
  - **Sequence Length**: A sequence length parameter is set to determine the length of input sequences for training.
  - **Token Indices**: The text is numericalized into token indices using the vocabulary. These indices represent the text in a format suitable for training a neural network.
- Sequence **Preparation**:
  - **Length Adjustment**: The token indices are adjusted so that their length is a multiple of the sequence length. This ensures that each batch of data is consistent and manageable.
- Get **Item**:
  - **Input and Target Sequences**: For a given index, the corresponding input and target sequences are generated. The input sequence is composed of a fixed number of token indices, and the target sequence is the same sequence shifted by one position. This setup allows the model to learn to predict the next token in a sequence.

- Explanation:
  - The `Vocabulary` class is responsible for managing the tokenization and indexing of the text, ensuring that only the most relevant and frequent tokens are included in the vocabulary. The `CustomDataset` class structures the text data into sequences of consistent length, ready for training the neural network. This structured approach ensures that the data is efficiently processed and fed into the model, facilitating effective learning and text generation.

3. Model Implementation

- Purpose:
  - The goal of this step is to design and implement the architecture of the neural network for text generation. This involves creating a model that can learn from the input sequences and generate coherent text.
- Model Architecture:

1. **Embedding Layer**:
   - **Purpose**: The embedding layer converts input token indices into dense vector representations. This transformation is crucial because it allows the model to work with continuous data rather than discrete indices.
   - **Padding Index**: The embedding layer is initialized with a padding index to handle padded sequences properly. This ensures that the padding tokens do not affect the learning process.

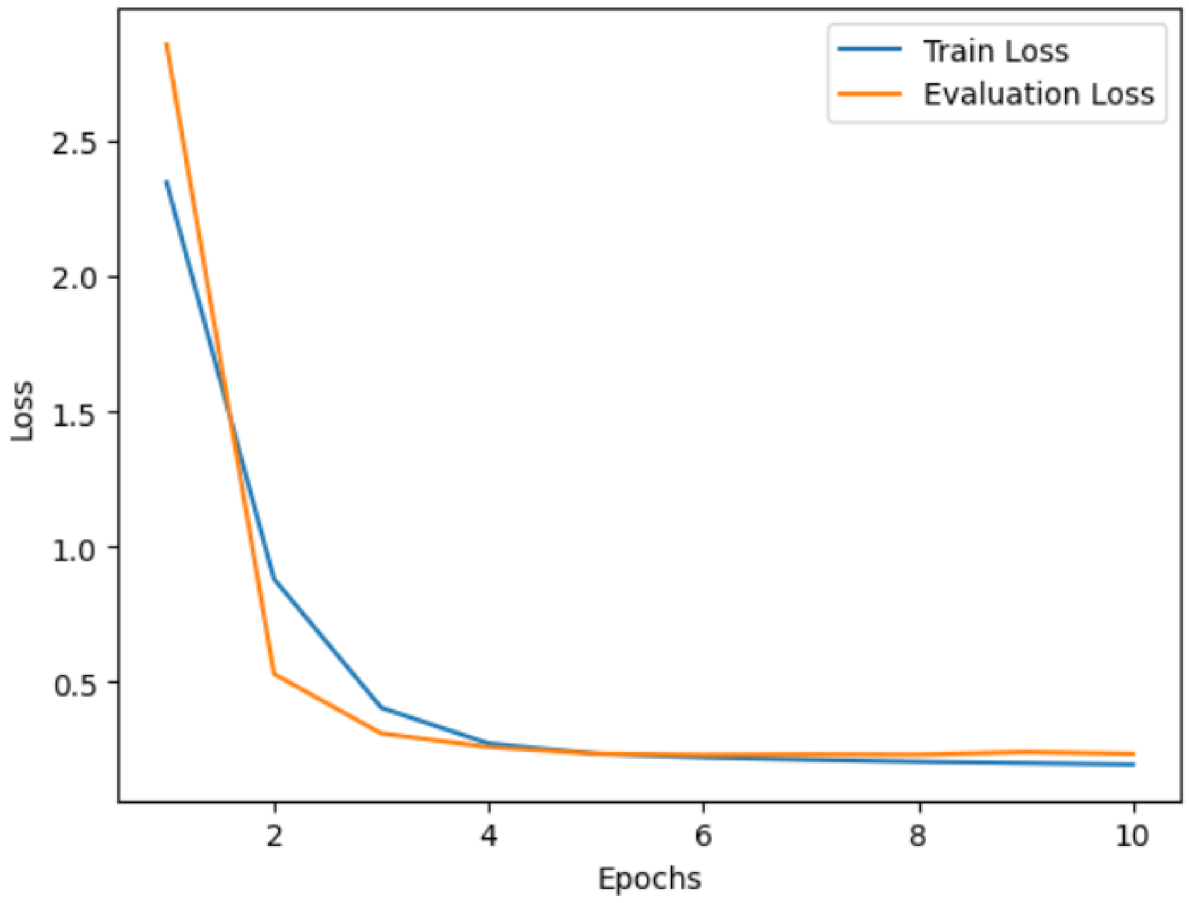
2. **LSTM Layer**:
    - **Purpose**: The LSTM (Long Short-Term Memory) layer processes the input sequences and captures the temporal dependencies and patterns within the data. LSTMs are particularly well-suited for sequence data because they can maintain information over long periods.
    - **Bidirectional**: The LSTM is bidirectional, meaning it processes the input sequence in both forward and backward directions. This helps the model capture context from both past and future tokens, improving its ability to generate coherent text.
    - **Hidden and Input Dimensions**: The LSTM layer is configured with specific input and hidden dimensions, determining the size of the input vectors and the internal state vectors.
3. **Fully Connected Layer**:
    - **Purpose**: The fully connected layer maps the LSTM outputs to the vocabulary size. This layer takes the hidden states produced by the LSTM and transforms them into a probability distribution over the possible next tokens in the sequence.
    - **Output Size**: The output size of this layer is equal to the vocabulary size, ensuring that each possible token has an associated probability.
4. **Forward Pass**:
    - **Purpose**: The forward pass defines how the input data flows through the model during training and inference.
    - **Embedding Transformation**: The input token indices are first passed through the embedding layer, transforming them into dense vectors.
    - **LSTM Processing**: The embedded vectors are then processed by the LSTM layer, which captures the sequential dependencies.
    - **Output Reshaping**: The outputs of the LSTM are reshaped to match the expected input size of the fully connected layer.
    - **Final Transformation**: The fully connected layer generates the final output, which is a distribution over the vocabulary for the next token in the sequence.

4. Training & Evaluating the Model

- Purpose:
    - The training process aims to optimize the neural network's parameters to minimize the loss function, thereby improving its ability to generate coherent text. This involves setting up the training loop, evaluating the model, and tracking its performance over multiple epochs.
- Process:

1. **Device Configuration**:
    - **Purpose**: Determines whether to use a GPU (if available) or CPU for training. This ensures that the model leverages available hardware resources for faster computation.
2. **Training Function**:
    - **Purpose**: Trains the model on the dataset by performing forward and backward passes through the network.
    - **Model Training Mode**: Sets the model to training mode, which enables gradient computation and dropout.
    - **Iteration over Training Data**: Iterates through batches of data from the training loader.
    - **Data Transfer to Device**: Transfers input and target data to the selected device (CPU/GPU).
    - **Zero Gradients**: Resets the gradients of the model parameters to zero before each forward pass to prevent accumulation from previous iterations.
    - **Forward Pass**: Computes the model's output for the current batch of inputs.
    - **Compute Loss**: Calculates the loss between the model's output and the actual targets.
    - **Backward Pass**: Computes the gradients of the loss with respect to the model parameters.
    - **Update Parameters**: Adjusts the model parameters based on the computed gradients using the optimizer.

    - **Loss Tracking**: Accumulates the loss for the current batch and computes the average loss for the epoch.
3. **Evaluation Function**:
    - **Purpose**: Evaluates the model's performance on the validation dataset without updating the model parameters.
    - **Model Evaluation Mode**: Sets the model to evaluation mode, disabling gradient computation and dropout.
    - **Iteration over Validation Data**: Iterates through batches of data from the validation loader.
    - **Data Transfer to Device**: Transfers input and target data to the selected device (CPU/GPU).
    - **Forward Pass**: Computes the model's output for the current batch of inputs.
    - **Compute Loss**: Calculates the loss between the model's output and the actual targets.
    - **Loss Tracking**: Accumulates the loss for the current batch and computes the average loss for the evaluation.
4. **Run Over Epochs**:
    - **Purpose**: Manages the training and evaluation process over multiple epochs to improve the model's performance iteratively.
    - **History Tracking**: Keeps a record of the training and evaluation losses for each epoch.
    - **Epoch Loop**: Iterates through a predefined number of epochs, calling the training and evaluation functions in each iteration.
    - **Print and Store Loss**: Prints and stores the training and evaluation losses for each epoch to monitor the model's progress.
5. **Plot History**:
    - **Purpose**: Visualizes the training and evaluation losses over epochs to provide insights into the model's performance and convergence.
    - **Plot Configuration**: Configures the plot with appropriate labels, legends, and axes.
    - **Loss Visualization**: Plots the training and evaluation losses to help diagnose issues like overfitting or underfitting.



6. **Show Predictions**:
    - **Purpose**: Demonstrates the model's ability to generate text by showing the original input sentences and the predicted next words.
    - **Prediction Extraction**: Extracts the predicted next word for each sequence by selecting the token with the highest probability.
    - **Original and Predicted Text**: Converts the input indices and predicted indices back to tokens and prints the original sentences along with the predicted next words.

```
-------------------------
Original Sentence: پرسپولیِ ، <UNK> پس از انتقال باشگاه راه آهن به ، NUM ورزشگاه کارگران انجام می شد . در شهریور
س ورزشگاه راه آهن را به
Predicted Next Word: عنوان
---------------------------------------------------------------------------------------
 -------------------------
Original Sentence: خلق می کند « <UNK> <UNK> » ؟ فاکس پاسخ می دهد که اهورامزدا با خلق موجودات آزاد ، نیروهای
که باز سؤال ایجاد می
 Predicted Next Word: شود
 ---------------------------------------------------------------------------------------
-------------------------
Original Sentence: به طور طبیعی و به راحتی در پا . <UNK> راه کنترل و حفاظت اشیاء گوناگون که متناسب نیست در نظر
یگاه داده های مشترک وجود
Predicted Next Word: دارد
---------------------------------------------------------------------------------------

-------------------------
riginal Sentence: میلیارد دلار که توسط دولت بریتیش کلمبیا طراحی و اجرا می شود  NUM <UNK> حمل و نقل ونکوور با
( در چند سال آینده تکمیل
Predicted Next Word: شد

 -------------------------
 Original Sentence: بخش پایینی گنبد _ حلقه ای که گنبد را استوار می کند _ آغاز شد ، به پایان رساندن طرح اجتناب ناپذیر ب
 ود . در
 Predicted Next Word: سال
 -------------------------------------------------------------------------------------
```

- Explanation:
    - **Device Configuration**: Ensures the model utilizes the best available hardware to speed up training and evaluation.
    - **Training Function**: Implements the core logic for training the model, including data handling, forward and backward passes, and parameter updates.
    - **Evaluation Function**: Provides a mechanism to assess the model's performance on validation data without affecting the learned parameters.
    - **Run Over Epochs**: Manages the iterative process of training and evaluating the model over multiple epochs, tracking performance improvements.
    - **Plot History**: Offers a visual representation of the model's learning progress, helping to identify trends and potential issues.
    - **Show Predictions**: Demonstrates the model's text generation capability by providing practical examples of input sequences and predicted outputs.