Aryan Neizehbaz – 400222112
5nd Assignment (LLM)

Data Loading and Preprocessing

The data loading and preprocessing phase is crucial for preparing raw text data into a format suitable for training a neural network. The following steps are undertaken:

1. **Data Directory Specification**: The directory containing the text files is specified (data_dir).
2. **File Reading and Text Normalization**: Each text file in the directory is read and normalized using the hazm library, which is designed for Persian language processing. The normalization process involves standardizing the text format and tokenizing the text into words.
3. **Corpus Creation**: The normalized text from all files is concatenated to form a single corpus. This corpus is a continuous string of text that serves as the basis for creating the dataset.

Dataset Creation

A custom dataset class, PersianWikipediaDataset, is created to handle the text data. The class inherits from torch.utils.data.Dataset and includes methods for managing the text data:

1. **Initialization**: The text data is split into words, and a vocabulary of unique words is created. Two dictionaries are generated: one mapping words to indices (word_to_idx) and the other mapping indices to words (idx_to_word).
2. **Length Calculation**: The length of the dataset is defined as the number of words in the text minus the sequence length.
3. **Item Retrieval**: The __getitem__ method returns a sequence of words (input) and the subsequent word (label) as tensors. This allows the model to learn to predict the next word given a sequence of preceding words.

Model Definition

The model used for text generation is a custom LSTM-based neural network, defined in the `TextGenerationModel` class. The model includes:

1. **Embedding Layer**: Converts word indices to dense vector representations.
2. **LSTM Layer**: Processes the embedded word vectors, capturing temporal dependencies between words.
3. **Fully Connected Layer**: Maps the LSTM outputs to the vocabulary space, providing probabilities for the next word.
4. **Hidden State Initialization**: The `init_hidden` method initializes the hidden states required by the LSTM.

Training

The model training process involves several steps:

1. **Hyperparameter Specification**: Key parameters such as vocabulary size, embedding size, hidden size, number of layers, batch size, number of epochs, and sequence length are defined.
2. **Optimizer and Loss Function**: The Adam optimizer is used to update model weights, and the CrossEntropyLoss function measures the prediction error.
3. **DataLoader Creation**: A DataLoader is created to handle batch processing of the dataset.
4. **Training Loop**: For each epoch, the model is trained on batches of data. The hidden states are reset for each batch. Mixed precision training is used to improve computational efficiency, involving loss scaling and gradient calculation.
5. **Model Saving**: The trained model's state dictionary is saved for future use.

Text Generation

The text generation process involves the following steps:

1. **Model Evaluation Mode**: The model is set to evaluation mode to disable dropout layers and other training-specific behaviors.
2. **Hidden State Initialization**: The hidden states are initialized for generating sequences.

3. **Sequence Generation**: Given a starting sequence of words, the model generates subsequent words iteratively. The generated text is formed by appending each predicted word to the input sequence.

Evaluation

The model's performance is evaluated using the ROUGE metric, which compares the generated text with reference text to assess the quality of text generation:

1. **Reference and Hypothesis Creation**: Random samples of text are taken from the dataset to create reference sequences. Corresponding sequences generated by the model serve as hypotheses.
2. **ROUGE Score Calculation**: The ROUGE scores are computed for the generated text, providing measures of recall, precision, and F1-score for unigrams (ROUGE-1), bigrams (ROUGE-2), and longest common subsequence (ROUGE-L).

ROUGE Scores Overview

ROUGE-1

- **Recall (r): 0.3075**
- **Precision (p): 0.1755**
- **F1-Score (f): 0.2229**

The model retrieves about 30.75% of the unigrams from the reference text, with 17.55% precision, indicating moderate performance in word-level matching.

ROUGE-2

- **Recall (r): 0.0461**
- **Precision (p): 0.0228**
- **F1-Score (f): 0.0305**

The low scores in bigram recall (4.61%) and precision (2.28%) suggest the model struggles with predicting consecutive word pairs accurately.

ROUGE-L

- **Recall (r): 0.2658**

- **Precision (p): 0.1516**
- **F1-Score (f): 0.1926**

The model captures some structural patterns with 26.58% recall but has lower precision (15.16%), reflecting moderate effectiveness in generating structurally similar sequences.

Model Fine-Tuning

Model Preparation

- **Model**: DistilBertForMaskedLM (distilbert-base-uncased) was used for fine-tuning.
- **Layer Freezing**: Parameters in the first four layers of the transformer were frozen to focus training on the later layers, which helps in transfer learning by retaining pre-learned features.

Optimization

- **Optimizer**: AdamW optimizer was employed with a learning rate of 5e-5, applied only to the trainable parameters.
- **Gradient Scaling**: Gradient scaling using GradScaler and mixed precision training (autocast) were implemented to manage large-scale data efficiently and to prevent gradient underflow.

Training Loop

- **Epochs**: The model was trained for one epoch.
- **Device**: Training was conducted on a GPU to accelerate computation.
- **Data Handling**: The pretraining data loader was iterated over, and batches were moved to the GPU. The optimizer and scaler were used to handle the backward pass and parameter updates.

Model Saving

- The fine-tuned model was saved for future inference and evaluation.

Evaluation

Dataset and DataLoader

- **Test Data**: A subset (10%) of the pretraining text was used for evaluation.
- **Evaluation Dataset**: Created using a masked language model dataset compatible with the tokenizer.
- **DataLoader**: Batched data with a size of 16 for efficient evaluation.

Evaluation Procedure

- **Model**: The fine-tuned model was evaluated without training (in eval mode) to prevent gradient updates.
- **Token Decoding**: Predictions and references were decoded to readable text using the tokenizer.
- **ROUGE Scorer**: ROUGE scores (rouge1, rouge2, rougeL) were calculated to measure the overlap between the generated text and the reference text.

Results

ROUGE Scores

- **ROUGE-1**: 0.0517
- **ROUGE-2**: 0.0242
- **ROUGE-L**: 0.0517

Interpretation

- **ROUGE-1**: Indicates that approximately 5.17% of the unigrams in the reference text were successfully generated by the model. This reflects the model's ability to predict individual words correctly.
- **ROUGE-2**: Shows a bigram overlap of about 2.42%, suggesting that the model has more difficulty capturing consecutive word pairs accurately.
- **ROUGE-L**: The long sequence overlap score (5.17%) implies that while some structural patterns are captured, there is significant room for improvement in generating coherent and contextually accurate text.