Assignment 3
Building a Transformer-based Grammatical Error Correction System

ELL884

April 2, 2025

Overview

In this assignment, you will build a **Grammatical Error Correction (GEC)** system using BART and T5, Hugging Face Transformers, and PEFT (Parameter-Efficient Fine-Tuning) techniques. You will:

1. **Parse and prepare** data from M2 files using the provided parser.
2. **Tokenize, preprocess, and train** a model for GEC.
3. **Implement a correction (inference) function** to generate corrected text from input sentences.
4. Use BLEU and exact accuracy metrics on test data.
5. **Document** and present your findings, including error analyses and insights into model performance.

Important Constraint: You must implement everything from scratch except:

- The `M2Parser` class (already provided)

You can, however, read and learn from these components. The idea is to let you focus on the training pipeline, model usage, and the GEC-specific logic.

1 Data Preparation

1. **Understand the M2 Format**
   You are given an `M2Parser` class that can parse M2 files into parallel (source, target) data. Inspect the code to see how errors are annotated and how corrections get applied.
2. **Prepare Training/Validation Data**
   Use the parser on the provided train.m2 dataset

2 Model & Training Pipeline

Goal: Fine-tune a model for GEC in a parameter-efficient way (LoRA / PEFT).

1. **Models to experiment with**
   You will experiment with **BART-Large** and **T5-Base**. In case of memory issue try using LoRA
2. **Tokenizer & Model Loading**
   Load models from Hugging Face and fine tune it for the task. Keep all hyperparameters (batch size, learning rate, etc.) in a separate `GECConfig` or dictionary structure for clarity.

3. **Preprocessing**

   Implement a function that takes raw (source, target) texts and returns tokenized samples suitable for seq2seq learning. This function should handle **max length** constraints, padding, truncation, and creation of the `labels` field for training.

4. **Hyperparameter Tuning**

   Experiment with hyperparameters (learning rate, LoRA `r` value, batch size, etc.). Compare performance across these runs on your validation set.

5. **Documentation & Code Organization**

   Keep your code well-structured. For example:

   - A `GECConfig` class for hyperparameters.
   - A `GECorrector` class for the model loading, training, and inference methods.

3 Correction (Inference) Function

Implement a method like:

```
batch_correct(sentences: List[str]) -> List[str]
```

1. **Load the trained model** in evaluation mode.
2. **Tokenize** the input sentences (respecting `max_source_length`).
3. **Generate** corrected outputs with `model.generate(...)` (leveraging parameters like `num_beams`, `length_penalty`, etc.).
4. **Decode** the outputs (e.g., `tokenizer.batch_decode(...)`).

Challenge: Consider memory usage by chunking large input sets into smaller batches (batch size). Add a progress bar (like `tqdm`) to monitor progress on large input sets.

4 Evaluation

- Uses `batch_correct` to produce corrected sentences,
- Computes BLEU via `sacrebleu` and a simple exact-match accuracy.

(Optional): Investigate more advanced GEC metrics like **GLEU** or the official M^2 scorer.

5 Report & Deliverables

Your report must be extensive, contain analysis over both models and hyperparameters, and report the best parameters based on BLEU and exact match.

There will also be a Kaggle submission where you will have to submit a corrected.txt file (assignment zip contains example of it)

Additionally:

- **Code**: A well-organized repository or submission directory.
- **README**: Clear instructions for how to run training, inference, and evaluation (which arguments/commands to use).

6 Grading Criteria

Category	Weight
Report	30%
Kaggle	70%

7 Getting Started

1. **Set up environment**:

```
pip install torch transformers datasets sacrebleu peft tqdm pandas
```

2. **Read** the starter code to understand the `M2Parser`.
3. **Implement** your solutions in a new script or notebook, making sure to:
   - Use the provided M2 parser for training data.
   - Reuse the evaluation function for scoring.
   - Replace placeholders (`TODO` sections) with your own logic for model training, inference, etc.
4. **Train & Evaluate**:
   - Train your model on the training split of an M2 dataset.
   - Evaluate on validation/test sets.
5. **Submit** your code, trained model outputs (if feasible), and your final report.

Submission Checklist

- **Code**:
  - `main.py` (or notebook) with all training logic.
  - corrected.txt on Kaggle
- **Report** (PDF/Markdown).
- **Model files** (optional if large; can provide a link if needed).
- **Instructions** for reproducing your results (a `README` or similar). **- Must run on Kaggle or Colab Free Tier GPUs**