Report: Fine-Tuning Pre-Trained Models for Text Normalization

1. Introduction

1.1 Objective

Text normalization is a critical preprocessing step in many natural language processing (NLP) applications. It involves transforming text into a consistent and standard format, which is especially important when working with noisy or non-standard inputs. The objective of this task is to fine-tune pre-trained sequence-to-sequence models to automatically normalize text effectively, minimizing the need for manual corrections. The focus is to evaluate the capabilities of fine-tuned models in generalizing unseen data and producing accurate normalized text.

1.2 Dataset

The dataset used for this task consists of parallel text pairs, where each pair includes:

1. **`raw_comp_writers_text`**: The noisy input text, which contains formatting errors, inconsistencies, and other irregularities.
2. **`CLEAN_TEXT`**: The target normalized text, representing the desired output.

The dataset underwent several preprocessing steps to prepare it for fine-tuning:

- **Cleaning**: Removal of special characters, numbers, and redundant spaces.
- **Lowercasing**: Standardizing the text format.
- **Tokenization**: Preparing the text for input into the models.

This structured approach ensures the dataset is suitable for training and evaluating sequence-to-sequence models.

1.3 Model Selection

Three pre-trained sequence-to-sequence models were chosen for experimentation: **T5**, **BART**, and **mT5**. These models were selected due to their state-of-the-art performance on text generation and transformation tasks.

1. **T5 (Text-to-Text Transfer Transformer)**:
    - Designed specifically for sequence-to-sequence tasks.
    - Treats every NLP task as a text-to-text problem, ensuring adaptability across diverse tasks.

2. **BART (Bidirectional and Auto-Regressive Transformers)**:
   - Combines the strengths of bidirectional encoding and autoregressive decoding.
   - Excels in tasks requiring robust text generation, such as summarization and denoising.
3. **mT5 (Multilingual T5)**:
   - Extends the capabilities of T5 to multilingual settings.
   - Particularly suitable for tasks involving cross-lingual data or diverse linguistic patterns.

These models were fine-tuned on the preprocessed dataset to learn the mapping from noisy to normalized text. Their performance was evaluated on various metrics to identify the most effective approach.

---

2. Methodology

2.1 Dataset Preparation

The dataset utilized in this project comprises parallel text pairs, with each pair containing:

- **`raw_comp_writers_text`**: The noisy input text with formatting errors, inconsistencies, and other irregularities.
- **`CLEAN_TEXT`**: The corresponding normalized text, representing the desired target output.

Key Statistics

- **Number of Samples**: Approximately [insert the total number of samples].
- **Input Features**: `raw_comp_writers_text` (source), `CLEAN_TEXT` (target).

Preprocessing Steps

To prepare the dataset for fine-tuning, the following steps were performed:

1. **Text Lowercasing**: All text was converted to lowercase to reduce variability in representation.
2. **Removal of Special Characters and Numbers**: Unnecessary characters, such as punctuation marks (excluding those critical for structure) and digits, were removed to simplify the input.
3. **Tokenization**: Text was tokenized using model-specific tokenizers to prepare it for input into sequence-to-sequence models.

These preprocessing steps ensured that the dataset was clean, consistent, and ready for training.

2.2 Model Fine-Tuning

Models Used

The following pre-trained sequence-to-sequence models were fine-tuned:

1. **T5 (t5-small)**:
    - Designed for text-to-text tasks, ensuring compatibility with text normalization.
2. **BART (facebook/bart-base)**:
    - Excels at denoising tasks, making it a strong candidate for handling noisy text inputs.
3. **mT5 (google/mt5-small)**:
    - A multilingual variant of T5, chosen to explore robustness in multilingual scenarios.

Sequence-to-Sequence Task Alignment

The text normalization task inherently involves mapping an input sequence (`raw_comp_writers_text`) to a target sequence (`CLEAN_TEXT`). All selected models are well-suited for this sequence-to-sequence paradigm due to their architecture and pre-training objectives.

Tokenization Process

- **Tokenizer**: Each model's respective tokenizer (e.g., T5Tokenizer, BARTTokenizer).
- **Maximum Sequence Length**: Limited to 128 tokens to balance computational efficiency and context retention.
- **Padding and Truncation**: Applied during tokenization to ensure uniform input and output lengths across batches.

Training Settings

The models were fine-tuned using the following configurations:

- **Batch Size**: 8.
- **Learning Rate**: 5e-5, with a linear decay schedule.
- **Number of Epochs**: 3, to prevent overfitting while ensuring convergence.
- **Optimizer**: AdamW, which combines weight decay with Adam for better regularization.

The training process involved minimizing the cross-entropy loss between the predicted normalized text and the ground truth.

2.3 Evaluation Metrics

The fine-tuned models were evaluated using multiple metrics to ensure a comprehensive assessment of performance:

1. **BLEU (Bilingual Evaluation Understudy Score)**:
   - Measures the similarity between the predicted and target sequences based on n-gram overlap.
2. **ROUGE (Recall-Oriented Understudy for Gisting Evaluation)**:
   - ROUGE-1: Unigram overlap.
   - ROUGE-2: Bigram overlap.
   - ROUGE-L: Longest common subsequence.
3. **Word and Character Error Rates (WER, CER)**:
   - WER: Evaluates word-level differences between the predicted and target sequences.
   - CER: Measures character-level differences.
4. **Jaro-Winkler Similarity**:
   - Quantifies string similarity by considering common prefixes and transpositions.
5. **Exact Match (EM)**:
   - The percentage of predictions that exactly match the target sequences.
6. **Accuracy**:
   - The proportion of correctly normalized sequences.

These metrics provide a balanced view of lexical, structural, and semantic accuracy, ensuring a holistic evaluation of model performance.

---

3. Results

Training Summary

The fine-tuning process was conducted over three epochs for most models, with one extended run (six epochs) for BART. The training and validation losses steadily decreased, indicating effective learning for most models except `google/mt5-small`, which failed to converge.

Test Results

The table below summarizes the test results across the evaluation metrics:

Model	Accuracy	BLEU	ROUGE-1	ROUGE-2	ROUGE-L	Jaro-Winkler	WER	CER	Exact Match	Total Time (min)
t5-small	0.66	0.16	0.73	0.64	0.73	0.87	0.71	0.97	0.66	10
facebook/bart-base	0.72	0.15	0.79	0.64	0.79	**0.93**	0.58	0.81	0.72	17
google/mt5-small	0.00	0.00	0.00	0.00	0.00	0.41	1.00	1.36	0.00	23
facebook/bart-base*	0.71	0.14	0.79	0.63	0.79	0.93	0.49	0.78	0.71	32

** Six epochs were used instead of three.*

Notable Findings

1. **BART Performance**:
   - **Best Metrics**: `facebook/bart-base` achieved the highest scores in ROUGE-1 (0.79), ROUGE-L (0.79), and Jaro-Winkler (0.93).
   - **WER and CER Improvements**: The lowest WER (0.49) and CER (0.78) were observed with six epochs of BART, showcasing its robustness in handling noisy inputs.
2. **T5 Observations**:
   - The `t5-small` model demonstrated moderate performance across metrics, with ROUGE-L (0.73) and Jaro-Winkler (0.87) as its strongest scores. However, its WER (0.71) and CER (0.97) were higher than BART.
3. **mT5 Limitations**:
   - The `google/mt5-small` model failed to converge, yielding near-zero scores across most metrics. This highlights potential issues with model initialization, dataset compatibility, or training hyperparameters.

4. **Trade-offs**:
    - A slight trade-off between BLEU scores and CER/WER was observed. Models with lower CER/WER had slightly reduced BLEU scores, potentially indicating over-optimization for structural similarity at the expense of lexical precision.

---

4. Analysis

Strengths of the Approach

1. **Sequence-to-Sequence Alignment**: The chosen models (T5, BART) are well-suited for text normalization tasks due to their encoder-decoder architecture.
2. **Robustness of BART**: BART consistently outperformed T5 across metrics, demonstrating its strength in denoising tasks, particularly in sequence-to-sequence scenarios.

Limitations

1. **mT5 Performance**: The multilingual model did not converge, potentially due to its pre-training focus on a wider range of tasks and languages, making it less specialized for this task.
2. **BLEU Score Trade-off**: Despite high ROUGE and Jaro-Winkler scores, BLEU scores remained low, indicating room for improvement in capturing n-gram-level lexical matches.

Challenges Faced

1. **Preprocessing Impact**: Ensuring consistency during preprocessing was critical but challenging due to variations in text formats.
2. **Hyperparameter Optimization**: While the learning rate and batch size worked well for T5 and BART, they were insufficient for mT5.
3. **GPU Time Constraints**: Training models with larger configurations (e.g., `bart-large`, `mt5-base`) was limited by available computational resources.

Comparison to Expected Baselines

- **T5 Performance**: While T5 delivered acceptable results, it did not outperform BART, which aligns with expectations given BART's specialization in denoising.
- **BART as Baseline**: BART exceeded baseline expectations, particularly with extended training (six epochs), showcasing its adaptability to this task.

Custom Deep Learning Model for Text Normalization

1. Model Architecture

The model is built using a **Transformer Encoder** architecture, inspired by BERT, to tackle the text normalization task. The key components of the architecture are as follows:

- **Subword Tokenizer (BERT-based)**: The model uses BERT's WordPiece tokenization, which splits words into smaller subword units. This method allows the model to handle out-of-vocabulary words and rare tokens, improving its generalization capabilities by working with subword-level features instead of word-level features.
- **Embedding Layer**: The tokenized input is transformed into dense vector representations, capturing the semantic meaning of each token.
- **Positional Encoding**: Positional encodings are added to the embeddings to account for the order of tokens in the sequence, allowing the model to learn the relationships between words based on their position within a sentence.
- **Transformer Encoder**: A stack of multi-head self-attention layers and feedforward networks. This enables the model to capture long-range dependencies and contextual relationships between words in the input sequence, which is crucial for the task of text normalization.
- **Fully Connected Layer**: After passing through the Transformer encoder, the output is fed into a fully connected layer, which generates predictions for each token in the sequence.

The model was implemented in PyTorch and designed to handle variable-length sequences of tokens, with padding applied to ensure consistent input sizes.

2. Training and Evaluation

The model underwent training for 5 epochs, utilizing the Adam optimizer and Cross-Entropy loss. The validation set was used to monitor performance during training. Evaluation metrics such as **BLEU**, **ROUGE**, **Jaro-Winkler**, **WER**, **CER**, and **Exact Match** were employed to assess the model's performance on the test set.

3. Results

The model demonstrated promising results, but further evaluation is necessary to determine its full potential. Here is the table showing the results:

Metric	BLEU Score	ROUGE-1 Score	ROUGE-2 Score	ROUGE-L Score	Jaro-Winkler Score	WER	CER	Exact Match
Score	0.05	0.59	0.37	0.59	**80.26**	0.76	0.89	0.38

These results indicate that while the model performs reasonably well on certain metrics (such as ROUGE and Jaro-Winkler), there is still room for improvement in terms of BLEU, WER, CER, and Exact Match scores. The low BLEU score, in particular, suggests that the model may struggle with generating more accurate or fluent text compared to the reference.

4. Challenges and Data Requirements

While the model architecture shows promise, the limited dataset size constrains its ability to achieve optimal performance. A larger dataset would help improve the model's ability to generalize and handle rare or complex text normalization cases more effectively. Expanding the dataset is crucial to enhancing the model's robustness.

5. Next Steps

- **Data Expansion**: Increase the dataset size to improve generalization and performance.
- **Model Fine-Tuning**: Experiment with different hyperparameters, architectures, and optimization strategies.
- **Further Evaluation**: Continue evaluating the model across different datasets and additional metrics to ensure robustness.