CENG 442: Natural Language Processing
Final Exam

Faruk Kaplan & Mert Altekin
January 4, 2025

Abstract

This paper introduces a text summarization model using a Pointer-Generator Network with Coverage Mechanism. Preprocessing steps such as text extraction, cleaning, tokenization, and sequence filtering ensure data quality. The model employs a bidirectional LSTM encoder-decoder with attention and coverage layers to reduce repetition and enhance summary coherence. Key hyperparameters are tuned for efficiency within GPU limits. Evaluation with ROUGE metrics shows the model generates concise and coherent summaries, achieving ROUGE-1 (0.4363), ROUGE-2 (0.1674), and ROUGE-L (0.4137) scores. Results highlight the model's effectiveness in capturing essential content and structure from input text.

Text Summarization Report

The implemented code is accessible by opening the text_summarization.ipynb file

1. Dataset

We used dataset that contains below columns:

- text
- prediction
- prediction_agent
- annotation
- annotation_agent
- id
- metadata
- status
- event_timestamp
- metrics

Among these features, we only used 'text' and 'prediction' columns

Preprocessing

In here, we prepared the dataset for training by extracting, cleaning, and tokenizing text. It ensures the text data is consistent and formatted correctly, which is crucial for the model to effectively generate summaries.

Extract Text

The 'prediction' column stores summaries in a dictionary format ('score': 'text'), so we need to extract the text . We wrote a function that extracts the 'text' from the 'prediction' column if the entry is a non-empty numpy array.

We eliminate 'score' because we don't have multiple summaries with different score points, every row has exactly one summary and their scores are "1.0", so cleaning is important

Handle Special Tokens

We added start (<sos>) and end (<eos>) tokens to each predicted summary.

These tokens help the model understand sequence boundaries, a common practice in sequence-to-sequence tasks.

Text Cleaning

Even though we know that TensorFlow's Tokenizer will handle, we wanted to apply basic text cleaning procedure. It basically:

>Converts texts to lowercase.

>Removes special characters (preserving punctuation).

>Reduces multiple spaces to a single space.

It standardizes the text, which improves the consistency and performance of the model.

Tokenization

We initialized a tokenizer with a vocabulary size limit of 20,000 and sets <UNK> for out-of-vocabulary words. It basically transforms text into numerical representations, essential for model training.

Padding and Truncation

We padded and truncated sequences to ensure consistent lengths. At first we tried to use longest text length among rows but it turned out that it is too long, and our GPU memory isn't sufficient for that so we had to limit it.

Basically if the text length is smaller than the limit, we padded it; if it is bigger than the limit, we truncated it

Filtering Out Empty Sequences

We identified and retained rows where the input sequence is non-empty. Because the empty sequences can disrupt model training, so filtering them ensures data integrity.

2. Model Development

We defined a Pointer-Generator Network with Coverage Mechanism for text summarization. This approach combines:

1. Sequence-to-sequence (seq2seq) modeling.
2. Attention mechanism to enhance focus on relevant parts of the input.
3. Pointer-generator network to address the out-of-vocabulary (OOV) problem by allowing the model to copy directly from the input.
4. Coverage mechanism to mitigate repetitive generation by tracking past attention distributions.

Coverage Attention Layer

Covarage attention layer enhances the standard Bahdanau attention by incorporating past attention (coverage) to avoid redundant focus. The implementation is:

- Dense layers calculate attention scores based on encoder outputs, decoder hidden states, and accumulated coverage.
- Softmax is applied to derive attention weights.
- The attention weights are added to the coverage vector at each decoding step.

Pointer-Generator Distribution

It merges the vocabulary distribution and attention (copy) distribution to form the final output distribution. The mechanism works as follows:

- The pointer network computes (generation probability).
- If it is high, the model generates from the vocabulary; otherwise, it copies from the input.

Pointer-Generator Model with Coverage

The general architecture is

1. Embedding Layer: Encodes input tokens.
2. Encoder: An LSTM processes input sequences.
3. Decoder: An LSTM generates target sequences.
4. Coverage-Attention Layer: Computes context vectors and updates coverage.
5. Pointer Mechanism: Combines vocab and attention distributions to generate the final output.

And the process works as follows:

1. Encoder processes the input to obtain hidden states.
2. Decoder iteratively generates summaries using attention and pointer mechanisms.
3. Coverage is updated at each step to prevent repetitive generation.

Model Training

And finally, we trained our model

Data Splitting

Training (80%) / Validation (10%) / Test (10%) split ensures proper evaluation and prevents overfitting.

Loss and Accuracy Functions

Sparse categorical crossentropy used to track the errors between predicted and ground-truth tokens. Also token-level accuracy used for comparing predicted tokens to targets.

Training Loop

Gradient tape is used to track computations and it applies gradients during backpropagation. Then, each decoding step generates the next token while updating coverage. Then, loss and accuracy are computed at each step.

Hyperparameters

We set our hyperparameters as:

- Embedding Dimension: 128
- Encoder/Decoder Units: 256
- Learning Rate: 1e-3
- Batch Size: 32

Also, the model trains over multiple epochs, optimizing using Adam.

We wanted to increase embedding dimension and encoder/decoder units to capture more complex patterns but our GPU couldn't lift it and crashed everytime, so these are the higher hyperparams we could use

Evaluation

ROUGE-1 (0.4363)
ROUGE-2 (0.1674)
ROUGE-L (0.4137)

Analysis of Results

For ROUGE-1:

- Approximately 43.6% of the unigrams in the generated summaries overlap with those in the reference summaries.
- This score indicates a moderate level of word overlap. While it shows that the model is capturing a significant portion of relevant content, there's room for improvement in terms of capturing more key terms and ensuring comprehensive coverage of the source material.

For ROUGE-2:

- Around 16.7% of the bigrams in the generated summaries match those in the reference summaries.
- Bigram overlap is typically lower than unigram overlap because it captures more specific phrases and context. A ROUGE-2 score of 0.1674 suggests that the model has some ability to maintain contextual and sequential accuracy but may struggle with maintaining fluency and preserving the original meaning in phrases.

For ROUGE-L:

- Approximately 41.4% of the longest common subsequence between the generated and reference summaries is preserved.
- ROUGE-L is sensitive to the overall structure and coherence of the summary. A score of 0.4137 indicates that the model maintains a reasonable level of structural similarity to the reference summaries, suggesting that it can generate coherent and logically ordered content, albeit not perfectly.

Further Enhancements

For improve our network performance, below enhancements can be applied:

- More Data: We can expand our dataset with more diverse and high-quality summaries.
- We can paraphrase existing summaries or use back-translation to create synthetic data.
- We can use reinforcement learning with ROUGE reward optimization
- We can increase model complexity, layers, units etc.