Wikipedia Title Generation Using Sequence-to-Sequence Models

Ritam Mandal Roll:24CS60R63

April 13, 2025

1 Abstract

This report investigates neural sequence-to-sequence (seq2seq) models for abstractive text summarization, focusing on generating concise titles from Wikipedia-style articles. It details the preprocessing pipeline, model architectures (RNN-based and T5 transformer models), training optimizations, and evaluation using ROUGE metrics. The study compares four RNN configurations and fine-tuned/prompt-based T5 models, analyzing trade-offs in performance, efficiency, and decoding strategies.

2 Introduction

Automatic text summarization is critical for applications like content recommendation and information retrieval. This report explores abstractive summarization to generate concise article titles using seq2seq models. Objectives include designing an efficient preprocessing pipeline, implementing flexible model architectures, optimizing training, employing advanced decoding strategies, and conducting robust evaluation. The study compares RNN-based models with transformer-based T5 models, highlighting their strengths and limitations.

3 Text Preprocessing

The preprocessing pipeline is tailored for title generation, balancing noise reduction with semantic preservation. For article texts, the process includes:

- **Lowercasing**: Converts text to lowercase for consistency.
- **Character Filtering**: Removes non-alphanumeric characters except basic punctuation (periods, commas, question marks, exclamation points) using regular expressions to reduce noise while preserving sentence structure.
- **Tokenization**: Uses NLTK's word tokenizer to split text into words.
- **Minimal Stopword Removal**: Removes high-frequency words (e.g., "the", "and", "a") to reduce noise while retaining contextual stopwords for meaning.

- **Lemmatization**: Applies NLTK's WordNetLemmatizer to normalize words to their base forms, ensuring vocabulary consistency.

For titles, preprocessing is adjusted to retain all stopwords, preserving their semantic importance in concise phrases. Punctuation is kept for clarity. The vocabulary is built from both preprocessed texts and titles, using a low minimum frequency threshold of 0.0000007 (minimum count of 19) to include most words, resulting in a vocabulary size of 46,040 tokens. Rare words are mapped to an unknown token to limit model complexity.

Vocabulary Statistics:

- Vocabulary size: 46,040
- Minimum count threshold: 19

4 Model Architecture

Two seq2seq approaches are implemented: RNN-based models and T5 transformer models.

4.1 RNN-Based Seq2Seq Models

The RNN-based models use an encoder-decoder framework with four configurations:

- **Basic RNN + Greedy Search**: A bidirectional GRU encoder captures context, with a unidirectional GRU decoder generating tokens autoregressively, selecting the most likely token at each step.
- **Basic RNN + Beam Search**: Identical to the basic RNN but uses beam search to explore multiple sequences for higher-quality outputs.
- **RNN with Hierarchical Encoder and Dual Decoder + Greedy Search**: A hierarchical encoder processes text at word and sentence levels to handle long documents, paired with a dual-layer GRU decoder for enhanced generation, using greedy search.
- **RNN with Hierarchical Encoder and Dual Decoder + Beam Search**: Combines the hierarchical encoder and dual decoder with beam search for diverse sequence exploration.

All models incorporate dropout for regularization.

4.2 T5 Transformer Models

The T5 model, a transformer-based architecture, frames title generation as a text-to-text task using self-attention. It is evaluated in two settings:

- **Fine-Tuned T5**: T5-small is fine-tuned on a Wikipedia-style dataset for optimal performance.
- **Prompt-Based Flan-T5**: Flan-T5-base and Flan-T5-large are tested in zero-shot settings with handcrafted prompts for flexibility.

5 Training and Optimization

Training is optimized for efficiency and stability:

- **Mixed Precision**: Reduces memory usage and speeds computation.
- **Gradient Clipping**: Prevents exploding gradients.
- **Early Stopping**: Halts training when validation performance plateaus.
- **Learning Rate Scheduling**: Dynamically adjusts learning rates.

A custom dataset handles variable-length sequences, with padding and sorting to optimize batch processing. Decoding strategies include greedy search for speed and beam search for quality.

6 Evaluation

Models are evaluated using ROUGE metrics (ROUGE-1, ROUGE-2, ROUGE-L) to measure n-gram overlap between generated and reference titles. Generated titles are post-processed with capitalization for readability.

6.1 RNN Performance

Four RNN configurations are compared. Table 1 provides space for ROUGE scores (to be filled by the user).

Table 1: ROUGE Scores for RNN Models

Model	ROUGE-1	ROUGE-2	ROUGE-L
Basic RNN + Greedy Search	0.2393	0.0535	0.2393
Basic RNN + Beam Search	0.2395	0.0477	0.2395
ROUGE-1: Hierarchical RNN + Dual Decoder + Beam Search	0.1291	0.0202	0.1291
Hierarchical RNN + Dual Decoder + Greedy Search	0.1330	0.0238	0.1330

6.1.1 Time Taken

Dual Decoder + Hierarchihcal Encoder for 10 epochs and 5 patience with batch size 16 : 3489 seconds Basic Encoder + Basic Decoder for 10 epochs and 5 patience and batch size 32:2896 seconds

6.1.2 Optimizing Memory for Efficient Training

Implementing the `optimize_memory` function significantly enhanced training efficiency on Kaggle's GPU environment. By invoking `torch.cuda.empty_cache()`, the function

clears unused memory from PyTorch's CUDA memory allocator, freeing up GPU resources for subsequent operations. This is critical in Kaggle's constrained hardware setting, where memory fragmentation can lead to out-of-memory errors during large model training. Additionally, setting the environment variable `PYTORCH_CUDA_ALLOC_CONF` to `expandable_segments:True` enables PyTorch to allocate memory more dynamically, reducing overhead and improving allocation efficiency. Monitoring GPU memory usage through `torch.cuda.get_device_properties` and `torch.cuda.memory_reserved` provided visibility into resource availability, allowing for better batch size tuning and model scaling. These optimizations collectively ensured stable, resource-efficient training, enabling faster experimentation and higher model performance within Kaggle's resource limits.

6.1.3 Analysis of Beam Search vs. Greedy Search Performance

Table 1 presents the ROUGE scores for various RNN models, revealing an unexpected trend: models using beam search consistently achieve lower ROUGE-2 scores compared to those using greedy search. For the Basic RNN model, beam search yields a ROUGE-2 score of 0.0477, compared to 0.0535 for greedy search. Similarly, for the Hierarchical RNN with Dual Decoder, beam search results in a ROUGE-2 score of 0.0202, while greedy search achieves 0.0238.

This counterintuitive result may stem from several factors. Beam search, which explores multiple hypotheses to optimize sequence probability, can prioritize fluency over diversity, potentially generating summaries that miss key bigrams captured by ROUGE-2. Greedy search, by contrast, makes locally optimal choices, which may align better with the reference summaries in this dataset, preserving critical phrases. Additionally, the small margin in ROUGE-1 and ROUGE-L scores (e.g., 0.2395 vs. 0.2393 for Basic RNN) suggests that the evaluation dataset or model architecture may not fully leverage beam search's strengths, such as handling longer dependencies. Further analysis, including qualitative evaluation of generated summaries, is needed to confirm these hypotheses.

6.2 T5 Performance

6.2.1 Fine-Tuned T5

The T5-small model was fine-tuned on a Wikipedia-style dataset. Table 2 shows ROUGE scores for greedy and beam search decoding.

Table 2: ROUGE Scores for Fine-Tuned T5

Decoding	ROUGE-1	ROUGE-2	ROUGE-L
Greedy Search	0.8780	0.6778	0.8780
Beam Search	0.8719	0.6678	0.8719

Execution time was 1706.30 seconds. Greedy search slightly outperformed beam search, likely due to the task's preference for short, precise titles. Greedy search's deterministic selection aligns with the fine-tuned model's high-probability token assignments, while

beam search's exploration may introduce valid but less exact matches, reducing ROUGE scores.

6.2.2 Prompt-Based Flan-T5

Zero-shot performance of Flan-T5-base and Flan-T5-large was tested with handcrafted prompts. Tables 3 and 4 show results.

Table 3: ROUGE Scores for Flan-T5-Base

Prompt	ROUGE-1	ROUGE-2	ROUGE-L
"Generate a title..."	0.7547	0.5321	0.7547
"Create a concise..."	0.4983	0.3346	0.4966

Table 4: ROUGE Scores for Flan-T5-Large

Prompt	ROUGE-1	ROUGE-2	ROUGE-L
"Summarize into a title..."	0.6052	0.4017	0.6035
"Appropriate title..."	0.7649	0.5603	0.7649

Execution times were 92.94 seconds (base) and 207.36 seconds (large). The "Appropriate title" prompt performed best, indicating prompt sensitivity in zero-shot settings.

7 Conclusion

The preprocessing pipeline effectively balances noise reduction and semantic preservation, supporting robust title generation. RNN-based models offer flexibility, with hierarchical encoders and dual decoders improving long-text handling. T5 models, particularly fine-tuned T5-small, achieved the highest ROUGE scores (0.8780 ROUGE-1), while prompt-based Flan-T5 demonstrated competitive zero-shot performance with lower computational cost. Greedy search's success in T5 suggests that simple decoding can suffice for concise tasks. Future work could enhance RNNs with attention mechanisms, explore larger transformers, or optimize prompt engineering for zero-shot settings.