Project Report

Long Document Summarization: Augmenting Unlimiformer with Knowledge Graphs[1]

Patrick O'Callaghan (patocal) Sheel Sansare (ssansa2) Tristan Wang (aawang99)

To-Do List (Remove When Done)

1. Explain objective function we are optimizing during training.
2. Submit test results to Scrolls.
3. Email Tolu on how to present code that doesn't run on Colab.
4. Present key result that length of summary is strongly dependent on input: LD < KG + LD < KG. Explain why this is.
5. Upload models to Hugging Face.
6. Figures

6.1 Shakespeare image with KG / dramatis personae.

6.2 The Mirror and the Light (Hilary Mantel).

6.3 Sheel's KG.

6.4 Plot distribution of LD, KG, and summary sizes for the 3 splits.

6.5 Graph convergence of summary length (number of tokens) to 90 for LDs, 750 for combined, 800+ for KGs.

6.6 Training / loss and other graphs from the training.

6.7 Table of results comparing R1, R2, RL, BERTScore F1 for the 3 experiments. Bold the best performers.

1 Introduction

In this blog post, we explore how knowledge graphs (KGs) can be applied to improve the accuracy of the `unlimiformer` long-range transformer for the task of long document (LD) summarization.

2 Problem Statement

Augmenting large language models (LLMs) to handle long documents using retrieval-based methods is a highly active area of research. The key weakness of modern transformer-based LLMs is the token limit, or context window of attention. For instance, the context

[1] December 11, 2023

window of ChatGPT-3.5 is 4,096 tokens, while the average novel contains well over 100,000 tokens.

`Unlimiformer` [**bertsch2023unlimiformer**], a recent retrieval-based method for augmenting LLMs at the decoder level, is the first long-range transformer to support unlimited length inputs. The key innovation of `unlimiformer` is to create a datastore of encodings which correspond to each token in the original document, and use the k-nearest neighbors (k-NN) algorithm to select the k most relevant tokens in the datastore during decoding.

Since KGs store richer information than plain datastores, we predict that feeding KG relational data as tokens into `unlimiformer` can enhance the model's understanding of LDs, like how a *dramatis personae* section can aid a reader in understanding a complicated novel by highlighting key relationships between characters. Thus, we hypothesize that the KG-augmented `unlimiformer` model will produce more accurate LD summaries than the baseline `unlimiformer` model.

3 Methodology

We compare and contrast the LD summaries generated by 3 transformer-based LLM models. Firstly, we train the facebook/BART base model using the `unlimiformer` augmentation, which operates on the entire LD and employs the k-NN algorithm. Secondly, we repeat the previous exercise but with KGs as inputs instead of LDs. Thirdly, we repeat the previous exercise with string inputs of concatenated KGs and LDs (in this order).

4 Creating the KGs and Datasets

Our baseline dataset is the Hugging Face version of GovReport [**huang2021efficient**], a well-established LD summarization dataset with many practical applications. To generate the required datasets, we use REBEL [**huguet2021rebel**], a pre-trained, end-to-end relation extraction model that can be found on Hugging Face here[2], to perform named-entity recognition (NER) and relation extraction (RE).

GovReport

Why GovReport, and how is it used?

REBEL

We chose REBEL because it is end-to-end (it finds entities and relations at the same time), open-source, and easy to implement using Hugging Face. Also, according to the DocRED paper by Yao et al [**yao2019DocRED**], pretrained REBEL yields the best joint

[2]https://huggingface.co/Babelscape/rebel-large

entity and relation extraction (NER and RE) result compared with the benchmark among all models sampled, with a relation F1 score of 47.1[3].

Since the pre-trained REBEL model has a token limit, we chose to split the LD into 128-token chunks before extracting head-relation-tail triplets from each chunk. Next, we used NetworkX to assemble the triplets and form a directed KG. The code for our REBEL implementation can be found here[4].

Why 128 span length?

Why extract triplets (and not extract triplets typed)?

Alternatives to REBEL

Other means of performing NER and RE we considered include spaCy-LLM, DyGIE++, and LlamaIndex. spaCy-LLM[5] is a package that integrates LLMs into natural language processing (NLP) pipelines provided by spaCy, an industry-standard NLP library. In particular, its built-in `spacy.REL.v1`[6] component supports RE with both zero-shot and few-shot prompting, but relies on an upstream NER component for entity extraction.

DyGIE++ is an RE component that refines and scores text spans designed to capture both intra-sentence and cross-sentence context. We cloned the code from the official GitHub repository linked here[7] and attempted to replicate the process of training a model for RE, but were unsuccessful due to technical difficulties.

Finally, LlamaIndex, a framework for connecting data sources for LLMs, has a class called `KnowledgeGraphIndex`[8] which is compatible with FAISS, the datastore that `unlimiformer` uses to conduct k-NN searches of top-level hidden state encodings, which would simplify our task of NER and RE.

5 Training

Unlimiformer

Why unlimiformer, and what is it?

BART

How do we use BART for training?

Work to train models individually.

[3]https://paperswithcode.com/sota/joint-entity-and-relation-extraction-on-3

[4]https://github.com/patrickocal/unlimiformer/blob/main/rebel.py

[5]https://spacy.io/usage/large-language-models

[6]https://github.com/explosion/spacy-llm/tree/main/usage_examples/rel_openai

[7]https://github.com/dwadden/dygiepp

[8]https://docs.llamaindex.ai/en/stable/examples/index_structs/knowledge_graph/KnowledgeGraphDemo.html

5.0.1 Background on BART

BART, like other transformer-based models, is considered adept at handling structured inputs due to several key features of its architecture and design.

**Structured Inputs

Structured inputs refer to data that is organized in a predictable, often hierarchical manner, with clear relationships between different parts. This contrasts with unstructured data, like free-form text, where the organization and relationships are not as explicitly defined. Examples of structured inputs include:

- Databases or tables where data is organized in rows and columns. - XML or JSON data, where elements are nested and have defined relationships. - Knowledge graphs, where information is represented as entities and relationships (triples).

Why BART Handles Structured Inputs Well

1. **Self-Attention Mechanism**: BART's transformer architecture uses a self-attention mechanism, which allows it to consider the entire input sequence at once. This enables the model to understand relationships between different parts of the input, essential for structured data.

2. **Contextual Understanding**: BART can capture context from both left and right of each token in the input sequence. This bi-directional context is crucial for understanding structured inputs, where the meaning often depends on the surrounding elements.

3. **Layered Encoding**: The layered structure of transformers enables them to capture and encode different levels of abstraction, which is beneficial for understanding hierarchical and nested structures in the input.

4. **Pre-training on Diverse Data**: BART is pre-trained on a wide range of data, including structured formats. This pre-training helps it to learn patterns and structures that are common in various types of data.

5. **Flexibility in Input Representation**: BART can handle sequences with special tokens and delimiters, allowing it to adapt to different types of structured inputs. For example, it can process inputs where parts of the data are segmented or highlighted using special tokens.

6. **Adaptability to Task-Specific Structures**: With fine-tuning, BART can adapt to specific types of structured inputs relevant to a particular task, enhancing its ability to process and generate meaningful outputs based on that structure.

In summary, BART's ability to process and understand the entire input sequence contextually, along with its adaptability and pre-training on diverse data, makes it well-suited for handling structured inputs. This capability allows it to effectively process and generate outputs based on inputs like knowledge graphs, tables, or other structured data forms. We chose to use the beginning of sequence (BOS, '<s>') and end of sequence (EOS, '</s>') tokens to separate triples in our knowledge graphs (KGs) with the intent of aligning BART's understanding of sequence boundaries, this approach has specific implications:

1. **Clear Segmentation of Information**: Using BOS and EOS tokens to delimit triples in the KG makes each triple a distinct segment from the model's perspective. This is beneficial since we want the model to treat each triple as an independent unit of information

since we expect our GovReport KGs to be such that the relationships within triples contain key information.

2. **Facilitating Attention Across Segments**: This segmentation should help the model's attention mechanism focus on each triple individually, potentially enhancing the model's ability to capture the nuances of each relationship within the KG.

3. **Model Adaptation to Structured Inputs**: Given that BART is designed to handle structured text, using BOS and EOS tokens in this way could aid the model in better understanding and generating summaries based on the structured nature of KGs. It aligns with the model's pre-existing mechanisms for processing text.

4. **Potential for Contextual Integration**: While each triple is treated as a separate sequence, the overall structure still allows the model to integrate these segments contextually. The model can learn to understand the KG as a whole, even though it processes each triple individually.

5. **Efficient Processing of Smaller Units**: By breaking down the KG into smaller segments, the model might process each unit more efficiently, especially if the triples are concise and the relationships within them are straightforward.

In this context, the slower training times you observed might not be due to the tokenization strategy per se but could involve other factors such as the complexity of the relationships in the KGs, the adaptation of the model to this unique structuring of inputs, or other computational aspects related to how the BART model processes these inputs.

Your approach aligns with the design principles of transformer models like BART, which are adept at handling structured inputs. The key would be to ensure that the rest of your training pipeline, including data preprocessing and model fine-tuning, is optimized to leverage this structure effectively.

Appropriateness of the BART Model

When training our model, we chose to feed the relational data of our KGs as tokens into `unlimiformer`, as opposed to embedding the KGs as separate relations into vector space. We believe that our approach is more appropriate as it allows us to better utilize the the `unlimiformer` framework, while preserving as much of the KG structure as possible within the dataset.

6 Results

How did our model perform compared to the baseline? Explanation?

Why is the average summary 800 words and not 500 words?

Interpreting the performance differences between models trained on long documents (LD) and knowledge graphs (KG) based on the provided metrics involves considering what each metric measures and how that relates to the nature of the inputs:

1. **ROUGE Scores**: - **ROUGE-1 (LD: 23, KG: 40)**: This measures the overlap of unigrams (individual words) between the generated summary and the reference summary.

The substantially higher score for KG suggests that the KG-based model is better at capturing key content words. This could be because KGs, being structured and concise, might enable the model to focus on essential terms more effectively. - **ROUGE-2 (LD: 11.74, KG: 11.47)**: This metric evaluates bigram overlap, indicating how well the model captures phrases and specific content. The similar scores suggest that both models are nearly equally effective at capturing phrase-level information, though the LD model has a slight edge. - **ROUGE-L (LD: 14.7, KG: 17.7)**: ROUGE-L assesses the longest common subsequence, which reflects sentence-level structure and coherence. The higher score for KG indicates better preservation of sentence structure or flow from the KG inputs.

2. **BERTScore**: - **Precision (LD: 0.69, KG: 0.58)**: Precision measures how much of the content in the generated summary is relevant or present in the reference summary. The higher precision for LD implies that it might be better at generating content closely aligned with the reference, likely due to the richer context provided by the long document. - **Recall (LD: 0.52, KG: 0.57)**: Recall assesses how much of the reference summary is captured in the generated summary. The higher recall for KG suggests it is better at including essential points from the reference summary, possibly due to the distilled and focused nature of KGs. - **F1/Aggregated BERTScore (LD: 0.59, KG: 0.57)**: This balanced metric considers both precision and recall. The scores are quite close, indicating that overall, both models are similarly effective in terms of content relevance and coverage, though the LD model has a marginal advantage.

Given these results after 8000 training steps:

- The KG-based model excels in capturing key content words and maintaining sentence structure, likely due to the concise and structured nature of KGs. - The LD-based model, despite its lower performance in some ROUGE metrics, achieves higher precision in BERTScore, suggesting its strength in generating summaries that are more aligned with the content and style of the reference summaries. This might be due to the rich, contextual information available in long documents. - The marginal differences in ROUGE-2 and BERTScore/F1 indicate that both models have their strengths and weaknesses. The LD model seems to be better for generating precise content, while the KG model is more effective in covering essential information and maintaining structure.

In conclusion, each model has its advantages depending on the desired outcome of the summary: whether the focus is on precise content alignment (LD) or on covering key points and maintaining structure (KG). Continued training and further optimization could enhance the performance of both models, potentially narrowing these gaps.

7 Conclusion

Do we recommend using KGs for LD summarization?