Developing a superior text summarization dataset and creating text summarization models fine-tuning pre-trained models with it.

Code: https://github.com/seungjun-green/GSoC

Problem

Nowadays, many people are using internet-scraped data to fine-tune or train their models. While this can lead to robust models, it raises significant copyright concerns. Plus companies like Reddit and X(Twitter) hiked their API fees, making data scraping harder.

Project Overview: Addressing the Problem

To address this problem, I created a custom dataset, CSV files with 4 columns(id, prompt, document, summary) and 11K rows using PaLM API. There are a total of 13 different document types such as copywriting, chat history, medical documents and so on. On average, the ratio of the summary length to the document length is 0.19.

How I created prompts, documents and summaries.

1. Creating Prompts

For each document type I divided it into several categories and per categories generated 100 - 200 related terms using PaLM API. Then using those terms, constructed 100 - 200 prompts for each category, applying appropriate sentence structures.

For instance, if the document type is medical, categories could be operative notes, diagnostic notes, and treatment plans. For the 'operative notes' category, I could generate 100 related terms like 'Adrenalectomy Surgery' or 'Arthroscopy Surgery' by passing prompts such as "Write down 100 surgery names." to PaLM API.

Then generated 100 prompts like "Write an operative note for an {age_group} patient who had a {surgery_name}."

Repeating this whole procedure for all 13 document types, I generated around 11K prompts.

2. Generating Documents and Summaries

Using the PaLM API generated 11K documents by passing those 11K prompts. Then passed the "Summarize following text.\nText:{document}\nSummary:" prompt to the PaLM API to get summaries.".

About the custom dataset:
This custom dataset is better than most of publicly available text summary datasets like XSum and CNN-Dailymail in following two areas:

Summary Quality: The summaries in the CNN-Dailymail and XSum datasets are often too short to capture the full meaning of the text. The CNN-Dailymail summaries are more like extractive summaries, while XSum's are abstractive but still too short, often being just one sentence long. In contrast, my dataset has a summary-to-document length ratio of 19% and includes high-quality text generated by the PaLM API.

Content Diversity: While CNN/Dailymail and XSum documents are all in news, my dataset includes 13 types of documents such as news, copywriting, chat history, medical documents and so on. Text Summary models trained on this dataset will be able to do tasks that require a broader range of language styles, topics, or formats.

Creating text summarization models
Then I created text summarization models fine-tuning GPT-2 varying three unique loss functions(1. Pay same importance to document and summary part, 2. Pay summary part twice importance to document part. 3, Only pay importance to the summary part) and T5 with the custom dataset. Based on the Rouge-L score I concluded that GPT2 fine-tuned with loss function paying same importance to document and summary part yields the best performance among 4 models.

What code got merged

I achieved 100% of the original goal, and my work got merged to google internal docs.

And beside the project, here are some pull requests that got merged into **Keras-io** repo.:

1. https://github.com/keras-team/keras-io/pull/1325
2. https://github.com/keras-team/keras-io/pull/1375
3. https://github.com/keras-team/keras-io/pull/1377
4. https://github.com/keras-team/keras-io/pull/1380

Any challenges or important things you learned during the project

Creating custom loss functions for fine-tuning GPT-2 was the most challenging part of my GSoC project.

I wanted to test how varying the ratio of importance being paid to document and summary part when calculating loss(1. Pay same importance to document and summary part, 2. Pay summary part twice importance to document part. 3. Only pay importance to the summary part) impact the result of performance to fine-tuned GPT2 models.

To do this I have to create a function that creates a weight tensor. It identifies 3 parts in the given tensor(y_true): document, summary, and padding, then fills a weight matrix with specified weights (doc_weight for the document part, 1.0 for the summary part, and 0.0 for the padding part).

At the time, I didn't have much knowledge about the dimension of tensors in Keras API, nor did I have experience in manipulating tensors using TensorFlow. This lack of knowledge made the task especially challenging for me. However, thanks to this experience, I've gained the skills and confidence to apply what I've learned to future projects.