Report
Dhruv Sood
2021387

Assignment-4 Report

DATA LOADING AND PREPROCESSING

I have loaded the dataset as Reviews.csv where total rows were around 5 lakhs and 10 rows but since we were supposed to make a summariser I have removed other columns and kept Text and Summary. There is a lot of duplicate data so dropping it made it to 393579 and dropped nan values.

I have used a big contraction mapping to further expand the words for better semantic catch.I have used text cleaner and summary cleaner to further preprocess and clean the text and summary of the dataset.

STATS

Average word length is 37.8 words and average summary length is 6 therefore I have taken max length to be 100 and 20 respectively.

PADDING

This code snippet imports necessary libraries and sets up a language model for text generation using gpt-2, a popular pre-trained model by hugging face's transformers library.
First, it imports the required modules such as re for regular expressions, torch for pytorch, and classes like gpt2lmheadmodel and gpt2 tokenizer from the transformers library.
then, it checks if a cuda-enabled gpu is available and assigns the device accordingly.
The gpt-2 tokenizer is initialized with its default settings, and the pre-trained gpt-2 model is loaded. The model is then moved to the specified device (either gpu or cpu).
Next, the code adds special tokens to the tokenizer. It defines <pad>, <bos> (beginning of sequence), and <eos> (end of sequence) tokens, which are used to handle padding and delineate the start and end of sequences.
The tokenizer's embeddings are resized to match the vocabulary size, ensuring compatibility with the updated special tokens.

TRAINING MODEL

This code segment prepares text data for training a gpt-2-based summarization model. It first imports necessary libraries including pandas for data manipulation and torch's dataset and data loader for efficient handling of training data. The data is split into training and testing sets using a 75-25 split ratio; a custom dataset class, summary dataset, is defined to process the text data. it concatenates each cleaned text with its corresponding cleaned summary, separating them

with a space, and encodes them using the gpt-2 tokenizer. The tokenizer converts the text into input ids and attention masks, ensuring consistent length and proper padding/truncation.
for each item in the dataset, the __getitem__ method returns a tuple containing input ids and attention masks. Finally, training and testing datasets are created using this custom dataset class, and corresponding data loaders are instantiated to facilitate efficient batch processing during model training and evaluation. The data is loaded in batches of size 8 and shuffled during training to enhance model learning.

SUMMARY GENERATION
Further defines functions for training and evaluating a gpt-2-based model for text summarization. it initializes the optimizer with adam and sets the device to cpu for computation. The training function iterates over epochs, batches the data using the provided data loader, performs forward and backward passes, and updates the model parameters based on computed gradients. It tracks training losses and terminates early if the average loss falls below a threshold. The evaluation function calculates the average loss on the test data without updating model parameters. Overall, this code orchestrates the training and evaluation process, enabling the iterative improvement of the gpt-2 model for text summarization.

SUMMARY
{user_input} is the generated summary for the user input.
{summary}
Model Loading: I used GPT2LMHeadModel.from_pretrained and GPT2Tokenizer.from_pretrained to load the saved model checkpoint (checkpoint-500). I have the end-of-sequence (EOS) token ID from the model setup as the pad token ID for the tokenizer.
Synopsis Creation: I created the method generate_summary, which uses the loaded model to generate a summary after encoding a review text using the tokenizer. For beam search decoding, the summary is produced using particular parameters like num_beams, length_penalty, and repetition_penalty.
User Data Entry and Synopsis Creation: I asked the user to provide a review text input, used the generate_summary function to create a summary, and printed the resultant summary.

ROGUE SCORE GENERATION
Measures: "rouge1," "rouge2," and "rougeL"
Stemmer Employed: Verified
Use Case Examples
The user-provided real summary is {actual_summary}.
{predicted_summary} is the anticipated summary.

ROUGE scoring Calculation: I created a function called calculate_rouge_scores that accepts as inputs the actual and predicted sums, initializes a RougeScorer with the ROUGE metrics "rouge1," "rouge2," and "rougeL," and computes the ROUGE scores by using the RougeScorer's scoring method.
Use Case Example: I used the previously generated summary as the anticipated summary and asked the user for the actual summary, which is presumed to be provided by the user. Next, you

use the calculate_rouge_scores function to calculate the ROUGE scores, and you report the results.
The purpose of using the rouge (recall-oriented understudy for gisting evaluation) score in gpt-2 text summarization between predicted and actual summaries is to quantitatively evaluate the quality of the generated summaries. rouge measures the overlap between the predicted summary and the reference (actual) summary based on various metrics such as overlap of n-grams (rouge-n), longest common subsequences (rouge-l), and word overlap (rouge-w).
by computing rouge scores, one can assess how well the generated summary captures the key information present in the reference summary. a higher rouge score indicates better agreement between the generated summary and the reference summary, implying better summarization quality. This metric helps in objectively comparing different models or fine-tuning strategies and can guide model development by providing feedback on areas for improvement. Overall, rouge scores serve as a valuable tool for evaluating the effectiveness of gpt-2 models in generating accurate and informative summaries.