Advanced Text Summarization Using Fine-Tuned BART

Abdul Basit Ali

M.S. student in the Khoury College of Computer Science at Northeastern University

ali.abd@northeastern.edu

Abstract

Many times, users need to read through and understand documents and articles that may span many pages. When such is the case, it can be very helpful to be able to generate a summary of said document(s). This aids the user in being able to capture the main points in a concise period, while still absorbing all of the important information. To do so, we can employ a system that is automated and can summarize such texts quickly and accurately. This can be used in many scenarios such as academia, work-related documents, novels, articles, and so on.

Existing solutions can be limiting as they may overlook certain contextual relationships between words in a given sentence, and can be restrictive due to their rule-based nature. Instead, we can employ the Transformer architecture, which excels in such tasks. However, they also have some limitations such as the fact that can be computationally expensive. We also need to make sure the model does not overfit.

The solution is to fine-tune a pre-trained transformer model on a concise summarization dataset such as XSum which is comprised of BBC articles as well as smaller summaries. Such a dataset can help us fine-tune an existing transformer model such as BART(Bidirectional and Auto-Regressive Transformers) to fit our task of summarizing documents. This model is automated and can summarize lengthy texts quickly and accurately. Furthermore, it utilizes advanced NLP techniques to capture complex relationships between words and preserve the essence of the original text. We can evaluate the model's performance using the ROUGE score which can help us determine the quality of the generated summaries.

Model Training and Calibration Methodology

The primary focus of this project is the fine-tuning of a pre-trained BART model for the task of text summarization. BART is a good choice as it is a Sequence-to-Sequence model. This means that it is naturally good at summarization. Furthermore, the model is Bi-directional, thus it understands context forwards and backward which is useful for the task of summarization. Finally, we will train it on an A100 GPU as BART is a large model and will require extensive computing power and resources to effectively train.

We aim to train the model on the XSum dataset, which is an aggregation of news articles as well as their summaries. This is useful as it allows the model to learn how to summarize lengthy documents into concise, but accurate text.

Before any training is performed, it is essential to ensure that the dataset has been pre-processed and formatted correctly. This will ensure that the input passed to the model will be optimized and will help the model learn effectively.

The first step of preprocessing is to initialize the model and tokenizer for BART. So, we load the BART model from the 'facebook/bart-large' checkpoint as well as the tokenizer designed for BART. Notice, that we are using BART-large as we want to maximize performance and contextual under-

standing to generate the best summaries possible. Next, we load the XSum dataset and split it into various sets. 204,045 for the training set, 11,332 for the validation set, and 11,334 for the test set.

The next step is to perform preprocessing. First, we tokenize the documents and summaries using the BART tokenizer. Then, we set the max token length, for documents, this is 1024 and for summaries, it is 128. Finally, we employ truncation and padding which helps to standardize the input length for the model.

```
def preprocess_data(batch):
    input_encodings = tokenizer(batch['document'], max_length=1024, truncation=True, padding="max_length") # tokenize document field with truncation and padding
    target_encodings = tokenizer(batch['summary'], max_length=128, truncation=True, padding="max_length") # tokenize the summary field with truncation and padding
    # Return dict of the key val pairs
    return {
        'input_ids': input_encodings['input_ids'],
        'attention_mask': input_encodings['attention_mask'],
        'labels': target_encodings['input_ids']
    }
```

Figure 1: Preprocessing the input

Continuing, we process every batch using the tokenizer to prepare the input for the model. It is essential to do so as we need to split the dataset into manageable chunks for training. We also establish a summarization pipeline, that is, a pipeline that uses BART for summarization. We then show the summarization using a sample text from the dataset.

```
def batch_data(data_list, batch_size):
    # Array to store batches
    batches = []

    # Loop through length of the data with a step size set to the batch size
    for start in range(0, len(data_list), batch_size):
        end = start + batch_size # start and end idx for curr batch
        batches.append(data_list[start:end]) # Add slice of data
    return batches
```

Figure 2: Batch Processing the input

Finally, we define the data collator. We employ the DataCollatorForSeq2Seq for dynamic padding and batch creation. It is also important to notice that the collator is compatible with the model and tokenizer as the tokenizer formats the input data for the model; thus compatibility is a must.

All of these preprocessing steps help to optimally prepare the dataset. This ensures the model's input is standardized and formatted correctly and will help us ensure that training is successful.

Model Training and Evaluation

Pre-Training Evaluation

We will first evaluate the BART model's performance on the XSum dataset before it has been fine-tuned via the ROUGE scores. We will define an evaluation function to do so.

This function is designed to assess the performance of a text summarization model on a given dataset. It does so by comparing the model-generated summaries with the reference summaries using a user-specific evaluation metric. The first step it performs is generating batches for the source documents and their respective summaries. Because XSum is so large batching is necessary as discussed in the previous section. It then loops through all of the pairs of the text and summary batches, and for each batch, the tokenizer encodes the text so it is readable by the model. Here we need to perform functions such as padding and truncation to make sure that the tensor sizes are consistent. We can then use these to generate summaries. Then, each generated summary is converted from the token-IDS back to text so we can compare them to the reference summaries. Finally, we can evaluate the performance for the specified metric to calculate the correct scores.

```
def evaluate_summarization(model, tokenizer, data, evaluate_metric,
                           text_col="document", summary_col="summary",
                           batch_sz=16, device_name=device):
    """
    Evaluates a summarization model on a given dataset for a given metric
    """

    # Generate bateches of text and summary data for eval purposes
    text_batches = batch_data(data[text_col], batch_sz)
    summary_batches = batch_data(data[summary_col], batch_sz)
    total_batches = len(text_batches)

    # Go through every batch of text and its summaries
    for text_batch, summary_batch in tqdm(zip(text_batches, summary_batches), total=total_batches):
        # Encode the text
        model_inputs = tokenizer(text_batch, max_length=1024, truncation=True,
                                 padding="max_length", return_tensors="pt")

        # Generate summaries for every part of text in the batch
        generated_summaries = model.generate(input_ids=model_inputs["input_ids"].to(device_name),
                                             attention_mask=model_inputs["attention_mask"].to(device_name),
                                             length_penalty=0.8, num_beams=8, max_length=128)

        # Convert token IDs to text
        decoded_summaries = [tokenizer.decode(gs, skip_special_tokens=True,
                                              clean_up_tokenization_spaces=True)
                             for gs in generated_summaries]

        # Compare the two summaries to get the eval
        evaluate_metric.add_batch(predictions=decoded_summaries, references=summary_batch)

    return evaluate_metric.compute()
```

Figure 3: Evaluation Function

We then evaluate the BART model's performance on

the test set of XSum pre-training. We use the following metrics: Capturing the main concepts of the source text(ROUGE1), the model's ability to keep relevant phrases from the source text(ROUGE2), and maintaining the structure and essence of the source text in the generated summaries(ROUGEL/Lsum).

	rouge1	rouge2	rougeL	rougeLsum
BART Model	0.161824	0.029526	0.104006	0.104019

Figure 4: ROUGE Scores for BART-large on XSum Pre-Training

As we can see, the scores are fairly poor. The model is not generalizing well on the XSum dataset. We then move to the next step, which is fine-tuning BART-large on the XSum dataset.

Model Training

Training a large model such as BART-large is a very time-consuming process. Even with state-of-the-art GPU's such as NVIDIA's A100 tensor-core GPU training BART for a few epochs takes many hours. Thus, training such a model multiple times is not a feasible task. Even if time was not an issue, A100's are very computationally expensive and thus you can only use them for a few hours on cloud services. This is to say that fine-tuning the training arguments for the model was the most important task. We had to ensure that every parameter was tuned correctly to ensure a smooth training process with as little time and resources lost as possible. Many configurations were tried, and a final config was reached after multiple dry runs on 500-1000 steps.

There are many hyper-parameters, so we will discuss all the relevant ones. For the number of epochs we chose 5 as this is a balanced choice. We also want to ensure that overfitting does not occur as BART is already good at summarization tasks, thus we want to merely fine-tune it for XSum. Continuing, we found that 8 for the batch size achieves a

```
training_args = TrainingArguments(
    output_dir='bart-large-output',
    num_train_epochs=5,
    per_device_train_batch_size=8,
    per_device_eval_batch_size=8,
    warmup_steps=500,
    weight_decay=0.01,
    learning_rate=3e-5,
    logging_dir='./logs',
    logging_steps=10,
    evaluation_strategy='steps',
    eval_steps=250,
    save_steps=500,
    load_best_model_at_end=True,
    metric_for_best_model='loss',
    greater_is_better=False,
    fp16=True,
    gradient_accumulation_steps=8
)
```

Figure 5: Final Training Arguments

perfect balance between memory usage and efficient training. For the warm-up steps we chose 500 as this allows the model to gradually increase the learning rate and not learn too quickly. This helps prevent overfitting and lets the model learn the new task. We set the weight decay to 0.01 to prevent overfitting. Choosing an appropriate learning rate was very important. We had to ensure that the model did not learn too quickly and succumb to overfitting and not too slowly as this could cause underfitting. We chose 3e-5 as it is usually a balanced learning rate for fine-tuning larger models like BART. It helps the model learn at a decent rate and avoids overfitting or underfitting. The evaluation steps are set to 250 to ensure that the model is learning throughout the entire training process and to catch any problems that may arise as soon as possible. Finally, we enabled F16 as it allows for faster computation on the A100 like quantization, and gradient accumulation steps are set to 8 to provide a healthy balance between memory usage and an effective batch size.

We can see that the BART model is learning the XSum dataset well as indicated by the training and validation loss decreasing. We can also see that there is no underfitting or overfitting occuring as the gap between the training and val-

[6524/15940 4:54:44 < 7:05:31

Step	Training Loss	Validation Loss
250	3.422200	2.833241
500	0.431100	0.392566
750	0.426900	0.379132
1000	0.406100	0.373573
1250	0.383300	0.366785
1500	0.408500	0.361220
1750	0.385100	0.362809
2000	0.387400	0.358453
2250	0.385300	0.355964
2500	0.392200	0.354608
2750	0.372500	0.353193
3000	0.389800	0.351427
3250	0.367500	0.351335
3500	0.367300	0.354674
3750	0.346100	0.350479
4000	0.346800	0.350724
4250	0.358600	0.350130
4500	0.338000	0.349168
4750	0.362800	0.347652
5000	0.352500	0.346941
5250	0.347500	0.346406

Figure 6: Training and Validation loss over Time

idation loss is small. Thus, the model is well calibrated, and the hyper-parameters chosen are effective. We will now evaluate the trained model.

Post-Training Evaluation

We will use the evaluation function defined in figure 3 once again, this time to evaluate the fine-tuned BART model on the XSum test dataset.

	rouge1	rouge2	rougeL	rougeLsum
BART Model	0.430808	0.209807	0.355767	0.355706

Figure 7: ROUGE Scores for BART-fine-tuned on XSum Post- Training

We can see that there is a significant improvement in ROUGE scores after fine-tuning the BART-large model. Specifically, a significant improvement in capturing the main concepts of the source text as indicated by the ROUGE1 score. A significant improvement in the model's ability to keep relevant phrases from the source text as indicated by the ROUGE2 score. Finally, a significant improvement at maintaining the structure and essence of the source text in the generated summaries as can be seen by the ROUGEL and Lsum scores. Thus, fine-tuning the BART model on the XSum dataset was a success. We can also examine the heatmap present in figure 8 of the ROUGE scores to help us better interpret them.

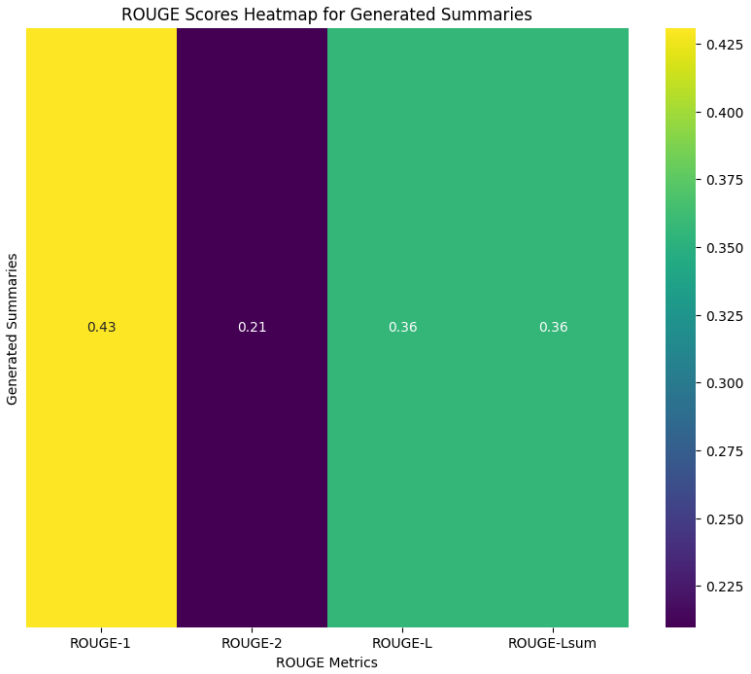

Figure 8: Rouge Scores Heatmap for Generated Summaries

We can examine that based on the color intensity as well as the score values, the heatmap indicates that the generated summaries have a strong overlap with the reference summaries in terms of singlular words as indicated by the ROUGE1 scores and a decent structural similarity as indicated by the ROUGEL and ROUGELsum scores.

We can now examine an example of a generated summary by the fine-tuned model.

In text format:

Document Snippet: Prison Link Cymru had 1,099 referrals in 2015-16 and said some ex-offenders were living

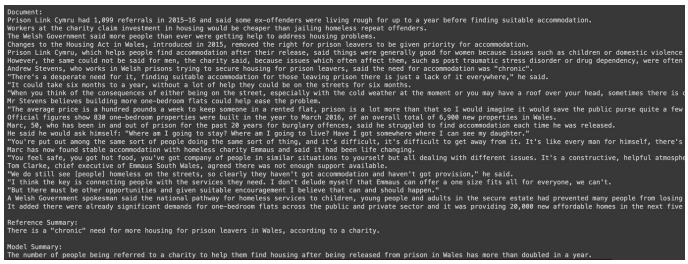

Figure 9: Generated Summary by the Fine-Tuned Bart Model

rough for up to a year before finding suitable accommodation. Workers at the charity claim investment in housing would be cheaper than jailing homeless repeat offenders. The Welsh Government said more people than ever were getting help to address housing problems. Changes to the Housing Act in Wales, introduced in 2015, removed the right for prison leavers to be given priority for accommodation. Prison Link Cymru, which helps people find accommodation after their release, said things were generally good for women because issues such as children or domestic violence were now considered. However, the same could not be said for men, the charity said, because issues which often affect them, such as post traumatic stress disorder or drug dependency, were often viewed as less of a priority. Andrew Stevens, who works in Welsh prisons trying to secure housing for prison leavers, said the need for accommodation was ”chronic”. ”There’s a desperate need for it, finding suitable accommodation for those leaving prison there is just a lack of it everywhere,” he said. ”It could take six months to a year, without a lot of help they could be on the streets for six months. ”When you think of the consequences of either being on the street, especially with the cold weather at the moment or you may have a roof over your head, sometimes there is only one choice.” Mr Stevens believes building more one-bedroom flats could help ease the problem...

Reference Summary: There is a ”chronic” need for more housing for prison leavers in Wales, according to a charity.

Model Summary: The number of people being referred to a charity to help them find housing after being released from prison in Wales has more than doubled in a year.

We can see that the model summary is well-formed, coherent, and touches on charity referrals, which is a relevant aspect present in the source document.

Conclusion

Fine-tuning the BART model on the XSum dataset helped it learn to summarize documents into effective summaries which accurately capture the theme(s) present in the source document. It also learned how to capture strong contextual relationships present in the source document as well as maintain the overall structure of the source text. The comparison between the before and after training ROUGE scores is a strong indicator of this as well as the example generated summary.

Preprocessing and formatting helped to prepare the dataset for training. Furthermore, correctly testing and tuning the training parameters configuration led to a well trained model that excels at generating coherent and accurate summaries.

References

Ding, N., Qin, Y., Yang, G. et al. Parameter-efficient fine-tuning of large-scale pre-trained language models. Nat Mach Intell 5, 220–235 (2023). https://doi.org/10.1038/s42256-023-00626-4

Howlader, Prottyee Paul, Prapti Madavi, Meghana Bewoor, Laxmi Deshpande, V.S.. (2022). Fine Tuning Transformer Based BERT Model for Generating the Automatic Book Summary. International Journal on Recent and Innovation Trends in Computing and Communication. 10. 347-352. 10.17762/ijritcc.v10i1s.5902.

Link to Code