Text Summarization

Rajkumar Conjeevaram Mohan

Introduction

In everyday life with a busy schedule, it is often difficult to spend time reading a lengthy document, especially for busy professionals who need just the gist. Having a concise form of the information that also preserves its meaning and the context will enable everyone to stay synchronized with the latest happenings around the world. There are two types of summarizations – Extractive and Abstractive; while the extractive type is based on selective sentences from the original text, abstractive type involves processing the complete text and generating a whole new text that is a summary of the original text. In this project, we have used T5 transformer architecture for summarizing text, and I have contributed mainly to optimize the code, and tune the hyperparameters given the boiler plate is available at Hugging Face.

Description of my work

Using one of my team member's Dataset codes, I wrote a completely custom training code from scratch. Also, I wrapped the original code with Class object so multiprocessing can be enabled. With such a huge dataset, working with a single worker is merely not possible to complete the training loop within a reasonable time frame. Even with the multiprocessing enabled, a single epoch takes over 3 hours to complete. I tuned some of the hyperparameters to minimize the overall loss.

Portion of my work

```
ts_df = pd.read_csv("data/test.csv")
tr_df = pd.read_csv("data/train.csv")
tr_df, vl_df, _, _ = train_test_split(tr_df, tr_df.highlights, test_size=0.3)
self.tr_size = tr_df.shape[0]
self.vl_size = vl_df.shape[0]

# raw_datasets = load_dataset("glue", "mrpc")
self.tokenizer = AutoTokenizer.from_pretrained(self.CHECKPOINT)
tr_tsdataset = TextSummarizationDataset(tr_df, self.tokenizer, MAX_INPUT_LEN, MAX_OUTPUT_LEN)
vl_tsdataset = TextSummarizationDataset(vl_df, self.tokenizer, MAX_INPUT_LEN, MAX_OUTPUT_LEN)
ts_tsdataset = TextSummarizationDataset(ts_df, self.tokenizer, MAX_INPUT_LEN, MAX_OUTPUT_LEN)
```

```python
# tokenized_datasets = tokenize_function(df)
data_collator = DataCollatorForSeq2Seq(tokenizer=self.tokenizer, model=self.CHECKPOINT)

self.tr_dataloader = DataLoader(tr_tsdataset, shuffle=True, batch_size=BATCH_SIZE,
                                collate_fn=data_collator, num_workers=N_WORKERS,
                                prefetch_factor=PREFETCH_FACTOR)
self.vl_dataloader = DataLoader(vl_tsdataset, batch_size=BATCH_SIZE,
                                collate_fn=data_collator, num_workers=N_WORKERS,
                                prefetch_factor=PREFETCH_FACTOR)
self.ts_dataloader = DataLoader(ts_tsdataset, batch_size=BATCH_SIZE,
                                collate_fn=data_collator, num_workers=N_WORKERS,
                                prefetch_factor=PREFETCH_FACTOR)
```

Custom Training loop:

```python
for epoch in range(self.num_epochs):
    model.train()
    loss_train = 0
    with tqdm(total=self.tr_size, desc=f"Epoch {epoch}") as progress_bar:
        for batch in self.tr_dataloader:
            batch = {k: v.to(device) for k, v in batch.items()}
            outputs = model(**batch)
            tmp_rouge = self.compute_metrics((outputs, batch['decoder_input_ids']))
            loss = criterion(torch.permute(outputs['logits'], [0, 2, 1]), batch['decoder_input_ids'])
            loss.backward()
            loss_train += loss.item()
            optimizer.step()
            lr_scheduler.step()
            optimizer.zero_grad()
            progress_bar.update(self.BATCH_SIZE)
            progress_bar.set_postfix_str("Train Loss: {:.5f}".format(loss_train/steps_train))

        # metric = load_metric("glue", "mrpc")
    model.eval()
    loss_val = 0
    with tqdm(total=self.vl_size, desc="Epoch {}".format(epoch)) as progress_bar:
        for batch in self.vl_dataloader:
            batch = {k: v.to(device) for k, v in batch.items()}
            with torch.no_grad():
                outputs = model(**batch)

            # logits = outputs.logits
            # predictions = torch.argmax(logits, dim=-1)
            loss = criterion(torch.permute(outputs['logits'], [0, 2, 1]), batch['decoder_input_ids'])
            loss_val += loss.item()
            progress_bar.update(self.BATCH_SIZE)
            progress_bar.set_postfix_str("Test Loss: {:.5f}".format(loss_val / steps_val))

        train_loss_batch = round(loss_train / self.BATCH_SIZE, 4)
        val_loss_batch = round(loss_val / self.BATCH_SIZE, 4)
        print(f"Epoch {epoch + 1} | Train Loss {train_loss_batch}, - Validation Loss {val_loss_batch}")
```

I thought the Trainer API would be a more concise and compact code to train, so I ended up re-using the team member's Trainer API code:

```
train_args = Seq2SeqTrainingArguments(
    output_dir='my_best_model',
    overwrite_output_dir=False,
    evaluation_strategy='epoch',
    learning_rate=self.LR,
    per_device_train_batch_size=self.BATCH_SIZE,
    per_device_eval_batch_size=self.BATCH_SIZE,
    # weight_decay=self.WEIGHT_DECAY,
    save_total_limit=3,
    num_train_epochs=self.N_EPOCH,
    gradient_accumulation_steps=self.GRADIENT_ACCUMULATION_STEPS,
    predict_with_generate=True,
    fp16=True,
    do_predict=True,
    # lr_scheduler_type='linear',
    half_precision_backend='auto',
    bf16_full_eval=False,
    dataloader_drop_last=True,
    dataloader_num_workers=self.N_WORKERS,
    # include_inputs_for_metrics=True,
)

trainer = Seq2SeqTrainer(
    model=self.model,
    args=train_args,
    train_dataset=self.train_dataset,
    eval_dataset=self.val_dataset,
    ##########################################################
    # For program testing
    # train_dataset=self.train_sample_dataset,
    # eval_dataset=self.val_sample_dataset,
    ##########################################################
    tokenizer=self.tokenizer,
    data_collator=self.data_collator,
    compute_metrics=self.compute_metrics,
)
```

Apart from the highlighted lines, there are also other lines I have written here. The highlighted lines are tweaked for quick experiments and optimization. For instance, I set the GRADIENT_ACCUMULATION_STEPS = 1 so that the parameters are updated after presenting the network with a mini batch. I personally believe using Mini batch is a good balance between SGD and Batch optimization methods.

The baseline model itself was computationally expensive, and I was unable to increase either the layer count or the attention heads to observe the difference in performance. The GPU usage for the baseline architecture:

```
ip-10-0-1-45  Sun May  7 02:52:04 2023  510.39.01
[0] Tesla M60 | 60°C,   4 % |  7654 /  7680 MB | ubuntu(7586M) gdm(3M)
```

P.S: The time zone in the screenshot is not EST.

Results

Most of my work involved tuning the hyperparameters of the model with the intention to enable it to generate both meaningful and concise summaries based on the texts it is fed with.

Optimized hyper-parameters:	
Learning Rate	1e-3
Batch size	30
Input Length	1024
Output Length	128
Gradient Accumulation Steps	1
N Parallel Calls	10
N Attention Heads	8
Query, Key, Value size	64
Epochs	30

Beyond 2nd – 3rd epochs with full dataset and 1024 input length, the Rouge score became almost flat, and was not improving. My understanding is, there are other hyperparameters such as number of attention heads, number of hidden layers, and hidden dimension of query, key, and value that need to be adjusted, perhaps increased, for the model to be able to perform well in generating summaries. Given the time constraints, we were unable to proceed further with tuning the hyperparameters.

Results at the end of 2nd epoch (full dataset):

Metrics	**1024 Input Length**	**256 Input Length**
Rouge Score	0.2013	0.054
Loss	1.05023	4.220

Results at the end of 2nd epoch (sample dataset):

Metrics	**1024 Input Length**	**256 Input Length**
Rouge Score	0.2424	0.2413
Loss	0.91646	0.633148

Sample output:

1. **Original Summary**: Experts question if packed out planes are putting passengers at risk. U.S consumer advisory group says minimum space must be stipulated. Safety tests conducted on planes with more leg room than airlines offer.

   Produced Summary: Tests conducted by the FAA use planes with a 31 inch pitch, a standard which on some airlines has decreased. Some airlines have 30 inches of space, while others offer as little as 28 inches. Some airlines offer as little as 28 inches.

Summary and Conclusion

I have primarily learnt how to use Transformers API by Hugging Face, and how importing a state-of-the-art model can be helpful for different NLP tasks. However, implementing the transformer from scratch would enable even deeper understanding of its mechanism, and I believe that would also put us in a better position to be to customize the model for our needs instead of using a pre-trained model.

In terms of improvements, I wish to implement the transformers architecture from scratch and observe its flow. Perhaps, that would allow me gain more in-depth knowledge of architecture and I am certain that would enable me to improve the metrics of even the current application. I would wish to implement T5 not only for text summarization, but instead combine many tasks such as question answering, text classification, etc., in one model, after all T5 can do this.

References

Alvin, T. P. (2022, March 21). *Introduction to text summarization with Rouge scores*. Medium. Retrieved May 5, 2023, from https://towardsdatascience.com/introduction-to-text-summarization-with-rouge-scores-84140c64b471