GenAI assignment report

On

Hate speech moderation by fine-tuning existing models

Submitted By:

Dharmanshu Singh(210398)

Under the mentorship of

Prof. Manisha Saini

Department of Computer Science and Engineering

School of Engineering and Technology

BML Munjal University, Gurugram

Speech Savior

Aaditya Vikram Agrawal, Dharmanshu Singh
School of Engineering and Technology
BML Munjal University, Haryana, India

Introduction

In the rapidly evolving field of Natural Language Processing (NLP), language models have emerged as powerful tools capable of generating human-like text. However, these models often require fine-tuning to perform specific tasks or to understand any particular domain more accurately. One such task is the generation of non-toxic content, a critical aspect in ensuring the safe and responsible use of AI technologies.

This report presents a project that aims to address this challenge by fine-tuning a FLAN-T5 model, a variant of the Transformer-based T5 (Text-to-Text Transfer Transformer) model, to generate less toxic content. The fine-tuning process leverages reinforcement learning, specifically Proximal Policy Optimization (PPO), and Parameter-Efficient Fine-Tuning (PEFT) using LORA.

The project also utilizes Meta AI's hate speech reward model, a binary classifier that predicts whether a given text is "hate" or "not hate". This reward model serves as a guide during the fine-tuning process, providing a signal to the FLAN-T5 model on the toxicity level of the generated content.

Through this project, we aim to demonstrate the potential of reinforcement learning in improving the quality of generated text by reducing its toxicity. We believe that this work will contribute significantly to the ongoing efforts to make AI technologies safer and more responsible.

Methodology:

The provided code follows a systematic approach to fine-tune a language model, specifically the Google FLAN-T5-base model, using reinforcement learning techniques. The goal is to generate summaries for dialogues that are less toxic and more factual. Here's a detailed breakdown of the methodology:

1. Environment Setup: The code begins by installing the required Python packages and libraries, such as PyTorch, Transformers, Datasets, PEFT (Parameter-Efficient Fine-Tuning), and TRL (Transformer Reinforcement Learning).

```
from transformers import pipeline, AutoTokenizer, AutoModelForSequenceClassification, AutoModelForSeq2SeqLM,
GenerationConfig
from datasets import load_dataset
from peft import PeftModel, PeftConfig, LoraConfig, TaskType

# trl: Transformer Reinforcement Learning library
from trl import PPOTrainer, PPOConfig, AutoModelForSeq2SeqLMWithValueHead
from trl import create_reference_model
from trl.core import LengthSampler

import torch
import evaluate

import numpy as np
import pandas as pd

# tqdm library makes the loops show a smart progress meter.
from tqdm import tqdm
tqdm.pandas()
```

2. Data Loading and Preprocessing: The "dialogsum" dataset from the Hugging Face repository is loaded, containing dialogues and their corresponding summaries. The dataset is filtered to include only dialogues with a length between 200 and 1000 characters. The dialogues are tokenized using the FLAN-T5-base tokenizer and wrapped with an instruction to summarize the conversation.

```
model_name="google/flan-t5-base"
huggingface_dataset_name = "knkarthick/dialogsum"

dataset_original = load_dataset(huggingface_dataset_name)

dataset_original
```

3. Model Architecture: The FLAN-T5-base model is a sequence-to-sequence transformer model based on the T5 architecture. It uses an encoder-decoder architecture with a shared weight matrix between the encoder and decoder. The FLAN-T5-base model is a variant of the T5 model pre-trained on a large amount of text data using a self-supervised language modeling objective.

4. Loading Pre-trained Checkpoint: The code loads the pre-trained "truocpham/flan-dialogue-summary-checkpoint" model and a pre-trained PEFT (Parameter-Efficient Fine-Tuning) checkpoint fine-tuned on the dialogue summarization task.

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

model_name1 = 'truocpham/flan-dialogue-summary-checkpoint'

model = AutoModelForSeq2SeqLM.from_pretrained(model_name1, torch_dtype=torch.bfloat16)
```

5. PPO Model Initialization: The pre-trained flan t5 model is converted into a PPO (Proximal Policy Optimization) model with a value head using the TRL library. The value head is an additional layer added to the decoder of the T5 model, which predicts the value function in the reinforcement learning setting. This model will be fine-tuned using reinforcement learning.

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer
from peft import PeftModel

peft_dialogue_summary_checkpoint = 'intotheverse/peft-dialogue-summary-checkpoint'

peft_model_base = AutoModelForSeq2SeqLM.from_pretrained("google/flan-t5-base")
tokenizer = AutoTokenizer.from_pretrained("google/flan-t5-base")

peft_model = PeftModel.from_pretrained(peft_model_base,
                                       peft_dialogue_summary_checkpoint,
                                       is_trainable=False)

print(f'PEFT model parameters to be updated:\n{peft_model.num_parameters()}\n')
```

```
ppo_model = AutoModelForSeq2SeqLMWithValueHead.from_pretrained(peft_model,
                                                               torch_dtype=torch.bfloat16,
                                                               is_trainable=True)

print(f'PPO model parameters to be updated (ValueHead + 769 params):\n{print_number_of_trainable_model_parameters
(ppo_model)}\n')
print(ppo_model.v_head)
```

6. Reward Model Setup: A pre-trained toxicity detection model (Facebook's RoBERTa-hate-speech-dynabench-r4-target) is loaded to evaluate the toxicity of generated summaries. The toxicity scores will be used as rewards during the reinforcement learning process.

```
toxicity_model_name = "facebook/roberta-hate-speech-dynabench-r4-target"
# toxicity_model_name = "unitary/toxic-bert"
# toxicity_model_name = "martin-ha/toxic-comment-model"
# toxicity_model_name = "DaNLP/da-electra-hatespeech-detection"
# toxicity_model_name = "JungleLee/bert-toxic-comment-classification"
toxicity_tokenizer = AutoTokenizer.from_pretrained(toxicity_model_name, device_map="auto")
toxicity_model = AutoModelForSequenceClassification.from_pretrained(toxicity_model_name, device_map="auto" )
print(toxicity_model.config.id2label)
```

```
from transformers import pipeline

device = "cpu"

sentiment_pipe = pipeline("sentiment-analysis",
                          model=toxicity_model_name,
                          framework="pt",
                          device=device)
reward_logits_kwargs = {
    "top_k": None,
    "function_to_apply": "none",
    "batch_size": 16
}

reward_probabilities_kwargs = {
    "top_k": None,
    "function_to_apply": "softmax",
    "batch_size": 16
}

print("Reward model output:")
print("For non-toxic text")
print(sentiment_pipe(non_toxic_text, **reward_logits_kwargs))
print(sentiment_pipe(non_toxic_text, **reward_probabilities_kwargs))
print("For toxic text")
print(sentiment_pipe(toxic_text, **reward_logits_kwargs))
print(sentiment_pipe(toxic_text, **reward_probabilities_kwargs))
```

7. Initial Model Evaluation: The code evaluates the toxicity of summaries generated by the pre-trained FLAN-T5-base model (before fine-tuning) on a test set, calculating the mean and standard deviation of the toxicity scores.

```python
def toxicity_evaluator(texts):
    inputs = toxicity_tokenizer(texts, return_tensors="pt", truncation=True, padding=True)
    with torch.no_grad():
        outputs = toxicity_model(**inputs)
    logits = outputs.logits
    probabilities = torch.softmax(logits, dim=1)
    toxic_probs = probabilities[:, 1].tolist()  # Probabilities of being toxic
    return {"toxicity": toxic_probs}
```

8. Training Setup: The PPO trainer and data collator functions are set up to handle the training process and data batching, respectively.

```
output_min_length = 100
output_max_length = 400
output_length_sampler = LengthSampler(output_min_length, output_max_length)

generation_kwargs = {
    "min_length": 5,
    "top_k": 0.0,
    "top_p": 1.0,
    "do_sample": True
}

reward_kwargs = {
    "top_k": None,
    "function_to_apply": "none",
    "batch_size": 16
}

max_ppo_steps = 10

for step, batch in tqdm(enumerate(ppo_trainer.dataloader)):

    if step >= max_ppo_steps:
        break

    prompt_tensors = batch["input_ids"]

    summary_tensors = []

    for prompt_tensor in prompt_tensors:
        max_new_tokens = output_length_sampler()

        generation_kwargs["max_new_tokens"] = max_new_tokens
        summary = ppo_trainer.generate(prompt_tensor, **generation_kwargs)

        summary_tensors.append(summary.squeeze()[-max_new_tokens:])

    batch["response"] = [tokenizer.decode(r.squeeze()) for r in summary_tensors]

    query_response_pairs = [q + r for q, r in zip(batch["query"], batch["response"])]
    rewards = sentiment_pipe(query_response_pairs, **reward_kwargs)

    reward_tensors = [torch.tensor(reward[not_hate_index]["score"]) for reward in rewards]

    stats = ppo_trainer.step(prompt_tensors, summary_tensors, reward_tensors)
    ppo_trainer.log_stats(stats, batch, reward_tensors)

    print(f'objective/kl: {stats["objective/kl"]}')
    print(f'ppo/returns/mean: {stats["ppo/returns/mean"]}')
    print(f'ppo/policy/advantages_mean: {stats["ppo/policy/advantages_mean"]}')
    print('-'.join('' for x in range(100)))
```

9. Reinforcement Learning Training Loop: The code enters a training loop where it generates summaries for dialogues in the training set using the current PPO model (FLAN-T5-base with value head). The generated summaries are evaluated for toxicity using the reward model, and the toxicity scores are used as rewards to update the PPO model's parameters using the TRL library's PPOTrainer.step() function.

```
batch_size = 20
compare_results = {}

df_batch = dataset["test"][0:batch_size]

compare_results["query"] = df_batch["query"]
prompt_tensors = df_batch["input_ids"]

summary_tensors_ref = []
summary_tensors = []

for i in tqdm(range(batch_size)):
    gen_len = output_length_sampler()
    generation_kwargs["max_new_tokens"] = gen_len

    summary = ref_model.generate(
        input_ids=torch.as_tensor(prompt_tensors[i]).unsqueeze(dim=0).to(device),
        **generation_kwargs
    ).squeeze()[-gen_len:]
    summary_tensors_ref.append(summary)

    summary = ppo_model.generate(
        input_ids=torch.as_tensor(prompt_tensors[i]).unsqueeze(dim=0).to(device),
        **generation_kwargs
    ).squeeze()[-gen_len:]
    summary_tensors.append(summary)

compare_results["response_before"] = [tokenizer.decode(summary_tensors_ref[i]) for i in range(batch_size)]
compare_results["response_after"] = [tokenizer.decode(summary_tensors[i]) for i in range(batch_size)]

texts_before = [d + s for d, s in zip(compare_results["query"], compare_results["response_before"])]
rewards_before = sentiment_pipe(texts_before, **reward_kwargs)
compare_results["reward_before"] = [reward[not_hate_index]["score"] for reward in rewards_before]

texts_after = [d + s for d, s in zip(compare_results["query"], compare_results["response_after"])]
rewards_after = sentiment_pipe(texts_after, **reward_kwargs)
compare_results["reward_after"] = [reward[not_hate_index]["score"] for reward in rewards_after]
```

10. Fine-tuned Model Evaluation: After the training loop, the code evaluates the toxicity of summaries generated by the fine-tuned PPO model on the test set, calculating the mean and standard deviation of the toxicity scores.

11. Results Comparison: The code compares the toxicity scores before and after fine-tuning, calculating the percentage improvement in mean and standard deviation.

12. Result Visualization: Finally, a pandas DataFrame is generated to visualize the differences in generated summaries and their corresponding toxicity scores before and after fine-tuning.

The overall objective of this methodology is to fine-tune the FLAN-T5-base sequence-to-sequence transformer model to generate less toxic and more factual summaries for dialogues by leveraging reinforcement learning techniques and a toxicity detection model as a reward signal.

Results and Conclusion

The project implements a powerful technique to fine-tune a language model for generating high-quality and non-toxic summaries of dialogues. By leveraging reinforcement learning methods and a toxicity detection model, the methodology aims to improve factual accuracy and reduce the toxicity of the generated summaries.

One of the key outcomes of this approach is the quantitative evaluation of the model's performance before and after the fine-tuning process. The code calculates the mean and standard deviation of the toxicity scores for the summaries generated by the initial pre-trained model and the fine-tuned model on a test dataset.

The results demonstrate a significant improvement in the toxicity scores after the fine-tuning process. The mean toxicity score decreases substantially, indicating that the generated summaries are less toxic and more appropriate. Additionally, the standard deviation of the toxicity scores also decreases, suggesting that the fine-tuned model produces summaries with consistently lower toxicity levels across different samples.

To better understand the impact of the fine-tuning process, the code provides a qualitative analysis by comparing the generated summaries and their corresponding toxicity scores before and after fine-tuning. This comparison is presented in a pandas data frame, allowing for a side-by-side examination of the changes in the generated summaries and their associated toxicity levels.

Furthermore, the code calculates the percentage improvement in the mean and standard deviation of the toxicity scores after fine-tuning. These quantitative metrics provide a clear measure of the effectiveness of the reinforcement learning approach in reducing the toxicity of the generated summaries.

Overall, the results demonstrate the power of the implemented methodology in fine-tuning language models to generate more factual and less toxic summaries for dialogues. By combining reinforcement learning techniques with a toxicity detection model as a reward signal, the code achieves significant improvements in the quality and appropriateness of the generated summaries.

Conclusions:

In this project, we as students have presented a methodology to address the challenge of hate speech and toxic language in dialogue summaries. The approach leverages reinforcement learning techniques in conjunction with a pre-trained toxicity detection model to fine-tune a language model for generating summaries that are not only factual and accurate but also significantly less toxic and more appropriate.

The results obtained through this study clearly demonstrate the efficacy of the proposed approach in mitigating toxicity levels in the generated summaries. The substantial reduction observed in both the mean and standard deviation of the toxicity scores after fine-tuning is a testament to the robustness and consistency of the approach across diverse samples.

For projects focused on hate speech moderation, we can argue that this methodology could prove

to be very effective. By integrating it into the existing pipeline, organizations could ensure that any dialogue summaries generated are not only informative and concise but also free from harmful or toxic language. This capability could be particularly beneficial in scenarios where dialogue summaries are presented to end-users, such as in chatbots, virtual assistants, or other conversational AI systems.
Furthermore, the quantitative nature of the improvement in toxicity scores provides a clear measure of the methodology's effectiveness, allowing organizations to objectively evaluate and monitor the performance of their hate speech moderation systems over time.

In conclusion, the presented methodology offers a powerful and practical solution for tackling the challenge of hate speech moderation in dialogue summaries. By combining state-of-the-art language models with reinforcement learning techniques and toxicity detection models, this approach can significantly improve the quality and appropriateness of the generated summaries, thereby ensuring a safer and more inclusive user experience.

Plagiarism

GENAI_assignment_report.docx.pdf

ORIGINALITY REPORT

6%	4%	4%	3%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Monash University Student Paper	1%
2	downloads.hindawi.com Internet Source	1%
3	www.csa.iisc.ac.in Internet Source	1%
4	Felicia Olmeta-Schult, Lauren Massa Segal, Sam Tyner, Twila Alexandra Moon et al. "NextGen VOICES: Research resolutions", Science, 2018 Publication	1%
5	Gurpreet Singh, Mansi Chaudhary, Jaspreet Singh, Nikita Madaan. "Parameter Design Approaches based on AI Techniques for Transformer Neural Network Optimization", 2023 3rd International Conference on Intelligent Technologies (CONIT), 2023 Publication	1%