LLM Instruction Backtranslation Assignment Report

Chong Chen

November 20, 2024

1 Introduction

1.1 Overview

This assignment focuses on fine-tuning a language model to reverse-engineer instructions from responses using the instruction backtranslation methodology. The primary goal is to implement a fine-tuning pipeline with limited computational resources to achieve efficient and effective training results.

1.2 Objective

The objective is to fine-tune a language model using low-rank adaptation (LoRA) to generate high-quality instructions from given responses. This involves setting up a training environment, preparing the dataset, and implementing the training pipeline.

1.3 Challenges

The main challenges include memory constraints on local hardware, ensuring compatibility between data formats and model configurations, and optimizing the training process to achieve the best possible results within resource limitations.

2 Model Selection and Environment Setup

2.1 Model Details

2.1.1 Base Model

The base model selected for this task is `tinyllama-1.1b`, a compact yet powerful model with 1.1 billion parameters and approximately 2.1GB in size. This model was chosen for its balance between performance and resource efficiency. The base model could be downloaded on `https://huggingface.co/Tourist99/tinyllama`.

2.1.2 Training Method

Low-Rank Adaptation (LoRA) was employed as the training method. LoRA allows for efficient fine-tuning by adapting only a small subset of the model's parameters, significantly reducing the computational and memory requirements.

2.2 Hardware Environment

2.2.1 GPU

The training was conducted on an NVIDIA RTX 3070 with 8GB VRAM. This GPU was selected for its balance between computational power and memory capacity, enabling efficient training within the constraints of local hardware.

2.2.2 System

The system used for training was a local PC running Ubuntu. The choice of a local system was driven by the need for flexibility and control over the training environment, despite the limitations in computational resources.

2.3 Justification for Approach

2.3.1 Why Local Training?

Local training was chosen due to several factors:

1. Limited access to external compute resources such as Google Colab or cloud-based services.

2. The manageable size of the dataset allowed for the effective use of memory-efficient techniques like FP16 precision and LoRA.

3. Local training provided greater control over the training environment, enabling fine-tuning of the model to specific requirements.

3 Training Pipeline

3.1 Data Preparation

3.1.1 Seed Data Source

The dataset used for fine-tuning was the **OASST1** dataset, available at `https://huggingface.co/datasets/OpenAssistant/oasst1`. This dataset contains pairs of instructions and responses, which are essential for the back-translation task.

3.1.2 Preprocessing

The dataset was preprocessed to transform it into `(response, instruction)` pairs suitable for backtranslation training. The preprocessing steps included:

- Tokenization using `AutoTokenizer` with a maximum sequence length of 512 tokens.
- Masking padding tokens with `-100` to ensure proper loss calculation during training.

The preprocessed data(used as seed data to train the backtranslation model) could be found on `data/seed/seed.jsonl`.

3.2 Training Details

3.2.1 Method

LoRA fine-tuning was employed with the following parameters:

- Rank: `r=16`.
- Alpha: `lora_alpha=32`.
- Dropout: `0.1`.
- Target modules: `q_proj`, `v_proj`.

3.2.2 Settings

The training settings were configured as follows:

- Batch size per GPU: 4 (effective batch size of 16 using gradient accumulation).
- Learning rate: `5e-4`.
- Epochs: 5.
- Precision: FP16.
- Checkpoints and logs were saved after each epoch.

3.2.3 Performance

The training performance metrics are as follows:

- Trainable Parameters: 2.25M ($\sim$0.204% of total model parameters).
- Training Time: 49 minutes for 5 epochs.
- Throughput: 2.93 steps/second.
- Final Training Loss: $\sim$5.07.

3.3 Implementation

3.3.1 Steps

The training pipeline was implemented in the following steps:

1. **Data Loading:** Loaded the JSONL dataset containing `instruction-response` pairs.
2. **Model Loading:** Initialized the base model with `AutoModelForCausalLM`, using FP16 precision.
3. **Training:**
    - Utilized gradient accumulation to achieve an effective batch size of 16.
    - Saved checkpoints after every epoch for resumption.
4. **Logging:** Generated comprehensive logs, including tokenization validation, training loss, and throughput metrics.

The trained backtranslation model could be downloaded on `https://huggingface.co/Tourist99/tinyllama-backtranslation-model-Myx`

3.4 Challenges

3.4.1 GPU Memory Limitations

The primary challenge was managing GPU memory limitations. This was addressed by using FP16 precision and LoRA, which reduced the trainable parameters from 1.1 billion to approximately 2.25 million.

3.4.2 Tokenizer Incompatibility

Dynamic addition of a `pad_token` was necessary to avoid token mismatch errors during preprocessing.

4 Instruction Generation

This section describes the process of generating instructions using the fine-tuned language model and the LIMA dataset. The goal was to create single-turn instruction-response pairs by filtering the dataset and generating new instructions based on given responses.

4.1 Dataset and Filtering

The LIMA dataset was utilized for this task. Each entry in the dataset consists of multi-turn conversations. To ensure single-turn examples, the dataset was filtered to retain only those entries where the conversation contained exactly two turns. The filtering process was implemented using the following logic:

- Load the dataset using the `datasets` library.
- Retain only those entries where the number of turns in the conversation equals two.

After filtering, 150+ single-turn conversations were selected for instruction generation.

4.2 Model Setup

The instruction generation model was created by fine-tuning a pre-trained TinyLLaMA model using LoRA-based techniques. The following steps were taken to set up the model:

1. Load the base model (`tinyllama-1.1b`) and the fine-tuned weights (`tinyllama-backtranslation-model-Myx`).

2. Merge the fine-tuned weights into the base model and unload redundant components to optimize memory usage.

3. Utilize the AutoTokenizer for text preprocessing, ensuring proper tokenization and handling of special tokens.

4.3 Instruction Generation Process

For each response in the selected dataset, a corresponding instruction was generated using the fine-tuned model. The following procedure was employed:

- Format the response as input in the form: `Response: <response>\\n\\nInstruction:`.
- Generate the instruction using the model's `generate` function with parameters set to encourage diversity and fluency (`temperature=0.7`, `top-p=0.9`, `top-k=50`).
- Extract the instruction from the model's output by removing the initial input text.

4.4 Example

4.5 Results and Analysis

The generated instruction-response pairs were evaluated for quality and relevance. Table 1 provides five examples of responses from real-world conversations alongside their corresponding generated instructions. Due to space constraints, the responses shown are truncated. The complete instructions and responses are available in the file `data/lima/lima_instructions.json`.

Analysis of Instruction Quality: The generated instructions were analyzed based on the following criteria:

1. **Relevance:** The generated instruction should be contextually aligned with the given response. For example:
    - In the second example, the response discusses the historical trend in CPUs, and the generated instruction, "What is the historical trend in CPUs and processors?" aligns well with the context.

2. **Clarity:** The instructions should be clear and concise. Most of the generated instructions fulfill this requirement, as seen in examples such as "How does the Tor client do the DNS requests?" which succinctly captures the essence of the response.

3. **Completeness:** While the instructions are generally relevant, some lack detailed context that could be derived from the response. For instance, in the fourth example, the response provides detailed information on command usage in Python, but the generated instruction focuses narrowly on comparing the advantages and disadvantages of using those commands without acknowledging the broader discussion.

4. **Originality:** The generated instructions occasionally mirror the wording of the responses too closely, reducing their originality. This is seen in the first example, where the instruction "Is the brain cells migrating or not?" could have been phrased more elegantly to avoid repetition.

General Observations:

- The model generally performed well in generating contextually relevant instructions from real-world responses.
- However, some generated instructions were overly broad or lacked specificity, indicating room for improvement in instruction refinement.
- Complex responses with detailed information, as in the fourth example, pose a challenge for the model, often leading to partial or oversimplified instructions.

Conclusion: The results indicate that the backtranslation model is capable of generating coherent and contextually appropriate instructions for most cases. While there are instances of suboptimal generation, the overall quality demonstrates the model's potential for tasks involving instruction-response pair generation. Future work could focus on fine-tuning the model to handle more nuanced and complex responses more effectively.

5 Conclusion and Future Work

5.1 Summary

The model `tinyllama-1.1b` was successfully fine-tuned for instruction backtranslation using LoRA. Resource constraints were effectively managed through

optimization techniques, and the model demonstrated the ability to generate relevant and coherent instructions from given responses.

5.2 Future Improvements

- Expand to multi-turn instructions for more complex backtranslation tasks.
- Experiment with larger datasets and models to improve generalization and performance.
- Investigate advanced techniques such as self-supervised learning and contrastive learning to enhance the quality of generated instructions.

References

- **Dataset:** OpenAssistant OASST1.
- **Paper:** "Self Alignment with Instruction Backtranslation" (arXiv:2308.06259).

Response	Generated Instruction
The question is relatively broad and one should take into account that the brain not only consists of neurons, but also glial cells (supportive cells) and pre-mitotic neuronal stem cells. Furthermore, ...	It is possible that the brain cells migrate, but the exact mechanism is not known. Is the brain cells migrating or not?
There is a general historical trend. In the olden days, memories were small, and so programs were perforce small. Also, compilers were not very smart, and many programs were written in assembler, so it was considered a good thing to be able to write a program using few instructions...	What is the historical trend in CPUs and processors?
Tor clients do not, in general, directly do DNS requests. When you open a connection through the tor network you usually ask your client (on its socks ports) to connect you to a hostname and port, say www.example.com:80...	How does the Tor client do the DNS requests?
“‘open“‘ and “‘start“‘ are command-interpreter things for Mac OS/X and Windows respectively, to do this. To call them from Python, you can either use “‘subprocess“‘ module or “‘os.system()“‘...	- “‘os.system(filename)“‘ - “‘subprocess.call([filename, ”_help”])“‘ - “‘os.system(”open ” + shlex.quote(filename))“‘ What are the advantages and disadvantages of using these commands in Python?
There are a few fundamental considerations: * Uncertainty is a part of the game. The referee and linesmen are the arbiters of the action. Instant replay would limit their authority...	When will the FIFA (Federation Internationale de Football Association) decide to allow instant replay?

Table 1: Sample generated instruction-response pairs.