INTELLIHACK 5.0 - INITIAL ROUND DELEGATES

Intellihack_DataDominators_TaskNumber03

Fine-Tuning Qwen2-0.5B: Detailed Documentation

Fine-Tuning Qwen2-0.5B: Detailed Documentation

1. Introduction

This document provides a comprehensive overview of the fine-tuning process of the Qwen2-0.5B model using Unsloth. It includes detailed reasoning for model selection, hyperparameter justifications, preprocessing steps, training methodology, and evaluation strategies. The primary goal of this fine-tuning was to adapt Qwen2-0.5B for answering questions based on markdown-based knowledge sources efficiently while maintaining performance within the constraints of Colab's limited resources.

2. Model Selection Reasoning

The **Qwen2-0.5B** model was selected based on the following criteria:

- **Compact yet powerful**: Qwen2-0.5B strikes a balance between efficiency and performance, making it ideal for fine-tuning in resource-limited environments. With 0.5 billion parameters, Qwen2-0.5B is a lightweight model that can be fine-tuned efficiently on a single GPU (Tesla T4 in this case). It strikes a balance between performance and resource requirements.
- **Performance**: Despite its smaller size, Qwen2-0.5B has shown competitive performance in various NLP tasks, making it a good candidate for fine-tuning.
- **Open-Source Availability**: Qwen2-0.5B is an open-source model, making it accessible for fine-tuning and customization.
- **4-bit Quantization Support**: Using Unsloth, this model can be quantized to 4-bit, significantly reducing memory consumption while retaining effectiveness.
- **Instruction-Tuning Capability**: Designed to handle structured Q&A and task-specific tuning effectively.
- **Seamless Integration with Unsloth**: Optimized for Hugging Face and Unsloth, ensuring a smooth fine-tuning process.

3. Training Process

The training process for the IntelliHack LLM Fine-Tuning Project involved loading the Qwen2-0.5B model with 4-bit quantization to optimize memory usage on the Tesla T4 GPU. LoRA (Low-Rank Adaptation) was applied to the model's attention layers, enabling efficient fine-tuning with fewer trainable parameters. The dataset, consisting of DeepSeek-related Markdown files, was preprocessed into 200-word chunks and tokenized with a sequence length of 128 tokens. The model was trained for *3 epochs* using a learning rate of *2e-4*, a

batch size of 2, and gradient accumulation over *4 steps* to simulate a larger batch size. Training was monitored using both training and validation loss, with the model saved at the end of each epoch. The final model was exported in both PyTorch and GGUF formats for deployment, ensuring efficient inference. This process balanced computational efficiency with effective domain adaptation, resulting in a fine-tuned model capable of generating coherent and contextually relevant text for DeepSeek-related tasks.

3.1 Dependency Installation

Before initiating the fine-tuning process, essential dependencies were installed to ensure compatibility with the training framework.

```
!PIP INSTALL TORCH==2.3.0+CU121 -F HTTPS://DOWNLOAD.PYTORCH.ORG/WHL/TORCH_STABLE.HTML

!PIP INSTALL UNSLOTH==2025.3.9

!PIP INSTALL TRANSFORMERS==4.48.3

!PIP INSTALL DATASETS==2.19.0

!PIP INSTALL NUMPY==1.26.4
```

3.2 Data Preprocessing

Dataset Preparation:

The dataset was prepared by collecting and preprocessing. The preprocessing steps included:

- Text Cleaning: Removing unnecessary whitespace and special characters.
- Chunking: Splitting the text into smaller chunks to fit within the model's sequence length limit.

Tokenizer Preparation:

The Qwen2 tokenizer was used to tokenize the text data. The tokenizer was configured to handle sequences of up to 128 tokens, with truncation and padding as needed.

Data preprocessing is a crucial step in fine-tuning to ensure that the model is fed clean and structured input data. Markdown files were read, cleaned, and chunked into manageable segments.

Read and Chunk Markdown Files

```
IMPORT OS

FROM DATASETS IMPORT DATASET

DEF READ_MD_FILES(DIRECTORY):

  DATA = []

  MD_FILES = ["DATASET.MD", "DEEPSEEKV3-EXPLAINED.MD"] # LIST OF MARKDOWN FILES

  FOR FILENAME IN MD_FILES:

    FILE_PATH = OS.PATH.JOIN(DIRECTORY, FILENAME)

    IF OS.PATH.EXISTS(FILE_PATH):

      WITH OPEN(FILE_PATH, "R", ENCODING="UTF-8") AS FILE:

        CONTENT = FILE.READ().STRIP()

        DATA.APPEND({"TEXT": CONTENT})

  RETURN DATA


DEF SPLIT_INTO_CHUNKS(DATA, CHUNK_SIZE=200):

  CHUNKED_DATA = [{"TEXT": " ".JOIN(ENTRY["TEXT"].SPLIT()[:CHUNK_SIZE])} FOR ENTRY IN DATA]

  RETURN CHUNKED_DATA
```

This process ensures:

- Data is structured correctly for tokenization.
- Text is segmented to prevent exceeding model context limitations.
- Markdown-based sources are formatted for efficient training.

3.3 Model Loading & Tokenization

Loading the model efficiently is essential to ensure fast and stable training. The tokenizer is initialized to handle the dataset effectively.

```
FROM TRANSFORMERS IMPORT AUTOTOKENIZER
FROM UNSLOTH IMPORT FASTLANGUAGEMODEL

MODEL_NAME = "QWEN/QWEN2-0.5B"
MODEL, TOKENIZER = FASTLANGUAGEMODEL.FROM_PRETRAINED(
  MODEL_NAME,
  MAX_SEQ_LENGTH=128,
  DTYPE=TORCH.FLOAT16,
  LOAD_IN_4BIT=TRUE
)

DEF TOKENIZE_FUNCTION(EXAMPLES):
  TOKENIZED = TOKENIZER(EXAMPLES["TEXT"], TRUNCATION=TRUE, PADDING="MAX_LENGTH",
MAX_LENGTH=128)
  TOKENIZED["LABELS"] = TOKENIZED["INPUT_IDS"].COPY()
  RETURN TOKENIZED
```

3.4 Training Configuration

The training process was set up using the Hugging Face Trainer API, with optimized parameters for performance and stability.

```
FROM TRANSFORMERS IMPORT TRAININGARGUMENTS, TRAINER
FROM DATASETS IMPORT LOAD_FROM_DISK

        TRAIN_DATASET = LOAD_FROM_DISK("/CONTENT/DRIVE/MYDRIVE/INTELLIHACK/DATASET/TRAIN")
        TEST_DATASET = LOAD_FROM_DISK("/CONTENT/DRIVE/MYDRIVE/INTELLIHACK/DATASET/TEST")

        TRAINING_ARGS = TRAININGARGUMENTS(
          OUTPUT_DIR="/CONTENT/QWEN2_FINETUNED",
          PER_DEVICE_TRAIN_BATCH_SIZE=8,
```

```
    PER_DEVICE_EVAL_BATCH_SIZE=8,
    LEARNING_RATE=5E-5,
    NUM_TRAIN_EPOCHS=3,
    WEIGHT_DECAY=0.01,
    EVALUATION_STRATEGY="EPOCH"
)

TRAINER = TRAINER(
    MODEL=MODEL,
    ARGS=TRAINING_ARGS,
    TRAIN_DATASET=TRAIN_DATASET,
    EVAL_DATASET=TEST_DATASET,
    TOKENIZER=TOKENIZER
)
```

3.5 Training Execution

The fine-tuning process was executed with the following command:

trainer.train()

This step initiates the fine-tuning, leveraging the pre-defined datasets, learning rate, and optimization settings.

4. Hyperparameter Justifications

Each hyperparameter was chosen based on best practices and the model's computational requirements:

- **Batch Size (8)**: Optimized for Colab's limited GPU memory.
- **Learning Rate (5e-5)**: A moderate learning rate to ensure stable convergence without drastic oscillations.
- **Epochs (3)**: Balances training time and model improvement while preventing overfitting.
- **Weight Decay (0.01)**: Regularization to mitigate overfitting.
- **Max Seq Length (128)**: Suitable for short-context question-answering tasks.

5. Fine-Tuning Methodology

- Pre-trained Qwen2-0.5B was leveraged for its strong foundational capabilities.

- Instruction Tuning was applied using markdown-based datasets to align responses to structured question-answering tasks.
- 4-bit Quantization reduced memory usage, enabling efficient training within Colab's environment.
- The Hugging Face Trainer API streamlined the supervised fine-tuning process.

6. Techniques to Enhance Reasoning Capabilities

7.1 LoRA for Efficient Fine-Tuning

LoRA was used to enhance the model's reasoning capabilities by allowing it to adapt to the specific domain of DeepSeek-related content. By fine-tuning only the low-rank matrices, the model retained its general language understanding while gaining domain-specific knowledge.

7.2 Chunking for Contextual Understanding

The text was split into chunks of 200 words, ensuring that the model had enough context to understand the relationships between different parts of the text. This helped improve the model's reasoning capabilities by providing it with more coherent input sequences.

7.3 FP16 Precision for Faster Training

Using FP16 precision allowed for faster training and reduced memory usage, enabling more efficient fine-tuning on limited hardware.

7. Evaluation

Post-training evaluation was performed using a dedicated test dataset, measuring model performance through:

- Loss Score on Evaluation Set
- Perplexity Calculation

```
EVAL_RESULTS = TRAINER.EVALUATE()

PRINT("EVALUATION LOSS:", EVAL_RESULTS["EVAL_LOSS"])
```

A lower loss value indicates better model generalization. Further refinements can be made based on the evaluation results.

8. Conclusion

The fine-tuning of Qwen2-0.5B was successfully executed with 4-bit quantization for efficient memory usage. The model was trained on markdown-based datasets and evaluated using a structured supervised fine-tuning approach. The use of LoRA and FP16 precision allowed for efficient fine-tuning on limited hardware, while the careful selection of hyperparameters ensured stable and effective training. The final model was saved in both PyTorch and GGUF formats, making it ready for deployment in various applications

Potential Future Enhancements:

- Expanding the dataset size to improve knowledge retention and generalization.
- Exploring RLHF (Reinforcement Learning from Human Feedback) to fine-tune response alignment further.
- Implementing additional post-training optimization techniques, such as LoRA-based adapters, for better task-specific performance.

This fine-tuning workflow demonstrates a structured approach to optimizing LLMs for specific use cases while ensuring computational efficiency.