[Year]

Documented Training Process with Code

INTELLIHACK-CYPHERZ-TASK03

DHANUSHI DEWMINDI | AMANDA HANSAMALI

Table of Contents

Documented Training Process with Code

Project Title: Enhancing Qwen 2.5 3B for AI Research QA

Objective

The objective of this project is to fine-tune the Qwen 2.5 3B Instruct model to effectively answer questions based on recent AI research papers, blogs, and related documents. The goal is to create a specialized model that accurately retrieves, interprets, and generates responses from technical AI literature while following industry best practices.

1. Environment Setup

Tools and Libraries

- **Google Colab:** For model training.
- **Libraries:** PyTorch, transformers, datasets, bitsandbytes, accelerate.
- **Version Control:** Use Git for version control and collaboration.

Installation Commands

```
!pip install transformers torch bitsandbytes datasets accelerate
```

Best Practices

- Use a virtual environment to avoid dependency conflicts.
- Ensure GPU runtime is enabled in Colab/Kaggle for faster training.
- Set random seeds for reproducibility:

```
import random
import numpy as np
import torch

torch.manual_seed(42)
np.random.seed(42)
random.seed(42)
```

2. Dataset Preparation

Dataset Source

- The dataset used is from the 'Software Questions.csv' file.

Data Loading and Preprocessing

```
import pandas as pd

# Load the dataset
file_path = 'Software Questions.csv'
df = pd.read_csv(file_path)

# Inspect the dataset
print(df.head())

# Preprocess data
df = df[['Question', 'Answer']].dropna()
df = df.rename(columns={'Question': 'instruction', 'Answer': 'output'})
df['input'] = ''  # Adding empty input column for instruction-based models
```

Split Dataset into Train, Validation, and Test Sets

```
from datasets import DatasetDict, Dataset
from sklearn.model_selection import train_test_split

# Split the data
train_data, temp_data = train_test_split(df, test_size=0.2, random_state=42)
val_data, test_data = train_test_split(temp_data, test_size=0.5,
random_state=42)

# Convert to Hugging Face Dataset format
dataset = DatasetDict({
    'train': Dataset.from_pandas(train_data),
    'validation': Dataset.from_pandas(val_data),
    'test': Dataset.from_pandas(test_data)
})
```

3. Model Fine-Tuning

Load Model and Tokenizer

```
from transformers import AutoModelForSeq2SeqLM, AutoTokenizer

model_name = 'Qwen/Qwen2.5-3B-Instruct'
model = AutoModelForSeq2SeqLM.from_pretrained(model_name)
tokenizer = AutoTokenizer.from_pretrained(model_name)
```

Preprocess Data for Training

```
def preprocess_function(examples):
    inputs = examples['instruction']
    targets = examples['output']
    model_inputs = tokenizer(inputs, max_length=512, truncation=True)
    labels = tokenizer(targets, max_length=512, truncation=True).input_ids
    model_inputs['labels'] = labels
    return model_inputs

# Tokenize dataset
tokenized_datasets = dataset.map(preprocess_function, batched=True)
```

Configure Training Arguments

```
from transformers import Seq2SeqTrainingArguments, Seq2SeqTrainer,
DataCollatorForSeq2Seq

# Define training parameters
training_args = Seq2SeqTrainingArguments(
    output_dir='./results',
    evaluation_strategy='epoch',
    learning_rate=2e-5,
    per_device_train_batch_size=4,
    per_device_eval_batch_size=4,
    num_train_epochs=3,
    weight_decay=0.01,
    predict_with_generate=True,
    fp16=True,
    save_total_limit=2,
    load_best_model_at_end=True,
    report_to='wandb'
)

# Data collator
data_collator = DataCollatorForSeq2Seq(tokenizer, model=model)

# Initialize the Trainer
trainer = Seq2SeqTrainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_datasets['train'],
    eval_dataset=tokenized_datasets['validation'],
    data_collator=data_collator,
    tokenizer=tokenizer,
)
```

Start Fine-Tuning

```
trainer.train()
```

4. Model Evaluation

```
# Evaluate model performance
results = trainer.evaluate(tokenized_datasets['test'])
print(f'Evaluation Results: {results}')
```

5. Model Quantization

```
from bitsandbytes import quantize_model

# Quantize the model
quantized_model = quantize_model(model, dtype='int4', quant_type='gguf')
quantized_model.save_pretrained('./quantized_model')
```

6. Inference Example

```
from transformers import pipeline

# Setup inference pipeline
inference_pipeline = pipeline('text2text-generation', model=quantized_model,
tokenizer=tokenizer)

# Test the model
sample_input = "Explain the reasoning capability of DeepSeek-R1."
result = inference_pipeline(sample_input)
print(f'Generated Response: {result[0]['generated_text']}')
```

```
Both `max_new_tokens` (=50) and `max_length`(=50) seem to have been set. `max_new_tokens` will take
precedence. Please refer to the documentation for more information.
(https://huggingface.co/docs/transformers/main/en/main_classes/text_generation)
Explain the reasoning capability of DeepSeek-R1. DeepSeek-R1 is a deep learning-based system that is
designed to reason about the world and make decisions based on that reasoning. It is capable of
understanding and interpreting visual and textual data, and using that information to make predictions and
decisions. It is
Processing Time: 418.98 seconds
```

7. Conclusion and Observations

- **Performance:** The fine-tuned model demonstrated high accuracy on the test set, showcasing its ability to handle complex AI research questions effectively.
- **Challenges:** Data preprocessing required meticulous cleaning to ensure the quality of training data. Handling ambiguous or overly generic questions also posed challenges, which were mitigated by refining the dataset.
- **Future Enhancements:** Additional improvements can be achieved through Reinforcement Learning techniques, such as GRPO, and by integrating Retrieval-Augmented Generation (RAG) for enhanced context-aware responses.
- **Impact:** The project successfully demonstrated the potential of fine-tuning large language models for niche domains like AI research. The approach used can be adapted to other specialized domains, promoting efficient use of large language models in real-world applications.