Fine-Tuning Llama 2 for Advanced Chatbot Development

This tutorial details fine-tuning the Llama 2 model for chatbots, using Weights & Biases and the Mintaka dataset, enhancing AI conversational abilities.

Mostafa Ibrahim

Created on January 19 | Last edited on February 23

Introduction

Chatbots, since their early days, have undergone quite a including customer service and healthcare, making them an integral tool in enhancing user experience and operational efficiency.

In this piece, we're going to look at fine-tuning Llama 2 to make our own prototype. Let's get started (after this space llama, of course).

What is Llama 2?

Llama 2 is a highly advanced language model with a deep understanding of context and nuances in human language. This makes it an ideal foundation for building advanced chatbots that can handle a wide range of conversational tasks with greater accuracy and relevance. Its ability to process and generate human-like text elevates chatbots to new levels of sophistication, allowing for more natural and effective interactions.

We've written a good deal about the Llama models, so if you'd like to read more, we recommend checking out:

Fine-Tuning LLaMa 2 for ...

How to Run LL...

In part 1, we prepped ou…

Exploring how SoftMax …

Weights & Biases

Weights & Biases (W&B) can play a crucial role in fine-tuning, providing a platform for tracking experiments, visualizing data, and managing models. When you're fine-tuning Llama 2, W&B helps in monitoring training progress, comparing different runs, and understanding model behavior. This is essential for optimizing the model's performance, ensuring that your advanced chatbot built on Llama 2 operates at its best. It's like having a smart assistant that keeps an eye on your model's training and helps you make informed decisions to improve it.

The Basics of Fine-Tuning Models for Chatbots

Fine-tuning in the context of AI and chatbots refers to the process of taking a pre-trained language model, like Llama 2, and adjusting it further to suit specific needs or tasks.

This model has already learned a lot about language from a large dataset, but fine-tuning it on specific chatbot-related data helps it understand the nuances of conversational language. This process involves training the model on a smaller, task-specific dataset, allowing it to become more adept at handling the types of queries

1. **Enhanced Performance:** It adapts the model to specific conversational contexts, improving its ability to respond accurately and contextually in chatbot interactions.
2. **Domain-Specific Knowledge:** Fine-tuning allows the model to learn the nuances and jargon of specific fields or industries, making the chatbot more relevant and useful in specialized areas.
3. **Improved User Experience:** By being fine-tuned on relevant data, the chatbot can provide more engaging, accurate, and efficient responses, enhancing user satisfaction.
4. **Efficiency:** Fine-tuning optimizes the model's capabilities, ensuring it operates effectively without unnecessary processing of irrelevant information.

Comparison With Other Adaptation Methods

Fine-tuning Llama 2 can be compared with other adaptation methods like transfer learning, domain adaptation, and zero-shot learning:

1. **Transfer Learning:** Similar to fine-tuning, transfer learning involves adapting a pre-trained model to a new task. However, fine-tuning is more specific, involving minor adjustments, while transfer learning can involve significant changes to the model's layers and structure.
2. **Domain Adaptation:** This focuses on adapting a model to a new domain while maintaining performance on the original task. Fine-tuning, by contrast, often specializes the model more narrowly, potentially at the expense of its original breadth of knowledge.
3. **Zero-Shot Learning:** This approach aims to apply a model to tasks

Using Llama 2 for your chatbot brings cutting-edge AI capabilities to your application. Its vast training on diverse datasets equips it with a deep understanding of language nuances, making interactions more natural and effective.

Further on, by fine-tuning Llama 2, you can create a chatbot that not only responds accurately but also understands context and user intent, offering a superior conversational experience. This makes it an excellent choice for sophisticated and user-friendly chatbot solutions.

Advantages of Using Llama 2 Over Other Models

Using Llama 2 for chatbots has several advantages over other models:

1. **Advanced Understanding:** Thanks to its extensive training, Llama 2 has a deep grasp of language nuances, aiding in more accurate and context-aware responses.
2. **Flexibility in Fine-Tuning:** It can be effectively fine-tuned for specific domains or user needs, offering tailored conversational experiences.
3. **Scalability:** Llama 2 can handle a range of tasks from simple Q&A to complex dialogues, making it versatile for various chatbot applications.
4. **State-of-the-Art Technology:** As a cutting-edge model, Llama 2 incorporates the latest advancements in AI and natural language processing, ensuring a high-quality chatbot performance.

Selecting an Appropriate Datasets and Data Format

To streamline the fine-tuning process for our Llama model, we will format our dataset in a specific way. Each entry in the dataset will begin with the question, followed by the te Comment s briefly as possible:", followed immediately by its corresponding answer. This structured approach ensures that the model receives clear and direct context for each training instance, enhancing its ability to learn and subsequently generate more accurate responses.

```
Question + "Answer as briefly as possible:" + Answer
```

For this tutorial, we are shifting our focus to leverage the Mintaka dataset by AmazonScience, a comprehensive Multilingual question-and-answer compilation.

This dataset stands out for its extensive range of question-answer pairs in multiple languages, drawing from a diverse array of subjects. By incorporating Mintaka into our training regimen for the Llama model, we are significantly broadening its linguistic capabilities and deepening its understanding across various languages and cultures. Moreover, we will update the outdated data that the model had with new and updated data.

Note that to better utilize the learning capabilities of our Llama model to your personal favor, select a dataset tailored to impart new knowledge, ideally one that reflects specific, new, and relevant content, such as proprietary company information or personalized data. This method of training ensures the model acquires and adapts

```
!pip install -qqq bitsandbytes==0.39.0
!pip install -qqq torch==2.0.1
!pip install -qqq -U git+https://github.com/huggingface/transformers
!pip install -qqq -U git+https://github.com/huggingface/peft.git@42a
!pip install -qqq -U git+https://github.com/huggingface/accelerate.g
!pip install -qqq datasets==2.12.0
!pip install -qqq loralib==0.1.1
!pip install -qqq einops==0.6.1
!pip install wandb
```

Step 2: Importing Necessary Libraries

In this step, we're laying the groundwork for our project by importing essential libraries that will play a pivotal role throughout our code. Initially, we're bringing in 'pandas' and 'json' for data handling and manipulation. Additionally, key functions like 'BitsAndBytesConfig' and 'AutoTokenizer' from the Transformers library, among others, will be crucial for model optimization and text processing, setting the stage for efficient and effective chatbot development.

```
import pandas as pd
import json
import os
from pprint import pprint
import bitsandbytes as bnb
import torch
import wandb
import torch.nn as nn
import transformers
from datasets import load_dataset, Dataset
```

process using Kaggle, we are provided with two T4 GPUs for our model fine-tuning.

```
os.environ["CUDA_VISIBLE_DEVICES"] = "0,1"
```

Step 3: Importing Llama 2 Chat HF Model

The model variable is being assigned a string that represents the path to the 13-billion parameter Llama 2 model stored on Kaggle. You can also download the model straight onto your PC for future usage.

```
model = "/kaggle/input/llama-2/pytorch/13b-chat-hf/1"
MODEL_NAME = model
```

Step 4: Performing Quantization

bnb_config is set with parameters for quantization, reducing the model's memory footprint by representing weights in 4-bit precision. This setup can improve performance on supported hardware. It is worth noting that training such a heavy model as Llama 2 requires a great sum of resources, to make the training process easier, we will quantize the weights of the model so that the fine-tuning process fits better inside our provided GPUS.

```
bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16
)
```

```
tokenizer.pad_token = tokenizer.eos_token

model = prepare_model_for_kbit_training(model)
```

Step 5: Import Necessary Functions

```
import re
def get_num_layers(model):
    numbers = set()
    for name, _ in model.named_parameters():
        for number in re.findall(r'\d+', name):
            numbers.add(int(number))
    return max(numbers)

def get_last_layer_linears(model):
    names = []

    num_layers = get_num_layers(model)
    for name, module in model.named_modules():
        if str(num_layers) in name and not "encoder" in name:
            if isinstance(module, torch.nn.Linear):
                names.append(name)
    return names
```

Step 6: Apply the LoRA Configuration

The below code will apply a Parameter-Efficient Fine-Tuning (PEFT) technique called LoRA (Low-Rank Adaptation) to the Llama 2 model. Which works by introducing low-rank matrices into specific layers of a pre-trained model. Rather than updating all the parameters of the model, LoRA only adjusts these added matrices.

```
model = get_peft_model(model, config)
```

Step 7: Importing Our Dataset

Below we will import the dataset of our choice. In our case, we will utilize the COVID-19 QA dataset.

```
df = pd.read_csv("/kaggle/input/multilingual-question-answering-data
df.columns = [str(q).strip() for q in df.columns]

data = Dataset.from_pandas(df)
```

Step 8: Defining Our Prompt Format

```
prompt = df["question"].values[0] + ". Answer as briefly as possible
```

Step 9: Setting Our Model's Configuration

We are now setting up how our model will generate text. One key setting is max_new_tokens, which we've limited to 10. This restricts the model from creating up to 10 new tokens, effectively controlling the length of the output. Another important parameter is temperature, which we've set to influence the randomness of the text generation. A lower temperature means the output will be more predictable and less random, giving us tighter control over the variety of the generated text. These configurations are crucial in tailoring the model's output to our specific needs.

```
generation_config = model.generation_config
```

Step 10: Importing the Prompt and Tokenization Functions

```
# Function to generate answers
def generate_answer_before(prompt):
    device = "cuda"
    encoding = tokenizer(prompt, return_tensors="pt").to(device)
    with torch.no_grad():
        outputs = model.generate(
            input_ids = encoding.input_ids,
            attention_mask = encoding.attention_mask,
            generation_config = generation_config
        )
    full_output = tokenizer.decode(outputs[0], skip_special_tokens=
    answer = full_output.split(prompt)[-1]  # Assumes the answer fo
    return answer.strip()

def generate_answer_after(prompt):
    device = "cuda"
    encoding = tokenizer(prompt, return_tensors="pt").to(device)
    with torch.inference_mode():
        outputs = model.generate(
            input_ids = encoding.input_ids,
            attention_mask = encoding.attention_mask,
            generation_config = generation_config
        )
    full_output = tokenizer.decode(outputs[0], skip_special_tokens=
    answer = full_output.split(prompt)[-1]
    return answer.strip()
```

Step 11: Evaluating the Old Model

```
    per_device_train_batch_size=1,
    gradient_accumulation_steps=4,
    num_train_epochs=30,
    learning_rate=1e-4,
    fp16=True,
    output_dir="finetune",
    optim="paged_adamw_8bit",
    lr_scheduler_type="cosine",
    warmup_ratio=0.01,
    report_to="wandb"
)

trainer = transformers.Trainer(
    model=model,
    train_dataset=data,
    args=training_args,
    data_collator=transformers.DataCollatorForLanguageModeling(toke
)
model.config.use_cache = False
trainer.train()
```

Step 13: Loading Our New Fine-Tuned Model

```
model.save_pretrained("trained-model")

PEFT_MODEL = "/kaggle/working/trained-model"

config = PeftConfig.from_pretrained(PEFT_MODEL)
model = AutoModelForCausalLM.from_pretrained(
    config.base_model_name_or_path,
    return_dict=True,
    quantization_config=bnb_config,
model = PeftModel.from_pretrained(model, PEFT_MODEL)
```

Step 14: Configuring New Model

```
generation_config = model.generation_config
generation_config.max_new_tokens = 30
generation_config.temperature = 0.3
generation_config.top_p = 0.7
generation_config.num_return_sequences = 1
generation_config.pad_token_id = tokenizer.eos_token_id
generation_config.eos_token_id = tokenizer.eos_token_id
```

Step 15: Evaluating the New Fine-Tuned Model

```
# Generate post-training answers
post_training_answers = [generate_answer_after(question + "Answer as

# Log the summary table to W&B
summary_table = wandb.Table(columns=["Question", "Answer Before Tra
for question, pre_ans, post_ans in zip(questions, pre_training_answe
    summary_table.add_data(question, pre_ans, post_ans)

wandb.log({"Summary Table": summary_table})

# Finish the W&B run
wandb.finish()
```

Output Explanation

Here's our results, in a handy W&B Table:

The initial incorrect response to the question, "What is the seventh tallest mountain in North America?" was, "The seventh tallest mountain in North America is Mount Bona, located in the Canadian Rockies. It stands at an elevation of..." This answer has been revised and updated to reflect accurate information. The correct answer is "Mount Lucania, Alaska. Mount Lucania is the seventh tallest mountain in North America, with an elevation of approximately 17,700 feet." This updated information is based on the dataset used to train the model.

How To Get Even Better Results

To optimize the fine-tuning process, it's crucial to concentrate on training the model with data that significantly differs from its existing knowledge base. The dataset mentioned earlier is characterized by its focus on highly specialized topics, each data point honing in on a distinct niche. While this specificity adds value, it may slow the overall learning progress of the model. For ideal fine-tuning outcomes, the model should ideally be exposed to a vast array of data points within a particular topic or niche. This approach ensures a more comprehensive and in-depth learning experience, enabling the model to develop a nuanced understanding and expertise in specific subject areas, thereby enhancing its overall performance and accuracy in those domains.

Biases highlights the precision and care needed to tailor these advanced models to specific conversational needs.

In essence, the fine-tuning of Llama 2 is not just a technical exercise; it's a step towards creating more engaging, intelligent, and responsive chatbots that can significantly enhance user experiences. As we continue to harness and refine these powerful AI tools, we edge closer to a future where digital communication is as nuanced and effective as its human counterpart.

Tags: LLM, Articles, Fine-tuning, Experiment