Agenda

- Hugging Face
- Llama 2
- Fine-tuning
The AI community building the future.

The platform where the machine learning community collaborates on models, datasets, and applications.
Hugging Face

- Hugging Face is a community and data science platform
- Provides tools that enable users to build, train and deploy ML models based on open-source code and technologies
Access Tokens

User Access Tokens

Access tokens programmatically authenticate your identity to the Hugging Face Hub, allowing applications to perform specific actions specified by the scope of permissions (read, write, or admin) granted. Visit the documentation to discover how to use them.

Create a new access token

- Name: new_token
- Role: write
Create a new model repository

A repository contains all model files, including the revision history.

Owner
JackyYu

Model name
New model name

License
License

Public
Anyone on the internet can see this model. Only you (personal model) or members of your organization (organization model) can commit.

Private
Only you (personal model) or members of your organization (organization model) can see and commit to this model.
What is Llama 2?

Llama 2 is:
• a collection of pretrained and fine-tuned large language models (LLMs)
• ranging in scale from 7 billion to 70 billion parameters
• use the transformer architecture
• open source, free for research and commercial use
What is Transformers?

1. Encoding Component: A stack consisting of N encoders.
2. Decoding Component: A stack consisting of N decoders
3. Connections between them.

The encoder extracts features from an input sentence, and the decoder uses the features to produce an output sentence.

https://sebastianraschka.com/blog/2023/lm-reading-list.html
What is Fine-Tuning?

• Process of training models with a size exceeding 10 billion parameters can present significant technical and computational challenges.
• Gradient descent on weights to optimize performance on one task.

• What to fine-tune?
  • Full network
  • Readout heads
  • Adapters

Parameter efficient fine-tuning

Change the model “itself”

https://www.labellerr.com/blog/hands-on-with-fine-tuning-llm/
Quantized Low-Rank Adaptation (QLoRa)

QLoRa works by quantizing the LLM to 4-bit precision.

- Freeze the pre-trained model weights.
- Reduce the number of trainable parameters for downstream tasks.
- Add Low-rank Adapter weights.

Adapter allow the network to quickly adjust to new tasks without major modifications to the entire architecture.

https://community.analyticsvidhya.com/c/generative-ai-tech-discussion/what-is-qlora
Use Google Colab to fine-tuning Llama 2
Check if GPU is out of memory
If it is full, you need to empty VRAM:

del model
del pipe
del trainer
import gc
gc.collect()
gc.collect()

Or,

!sudo fuser -v /dev/nvidia*

!sudo kill -9 PID
1. Install required packages:

```
!pip install -q accelerate==0.21.0 peft==0.4.0
bitsandbytes==0.40.2 transformers==4.31.0 trl==0.4.7
```

• transformers: This library provides APIs for downloading pre-trained models.
• bitsandbytes: It’s a library for quantizing a large language model to reduce the memory footprint of the model, especially on GPUs.
• peft: This is used to add a LoRA adapter to the LLM.
• trl: This library contains an SFT (Supervised Fine-Tuning) class to fine-tune a model.
• accelerate: This library used to increase the inference speed of the model.
All the libraries you need

- `import os`
- `import torch`
- `from datasets import load_dataset`
- `from transformers import (AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig, HfArgumentParser, TrainingArguments, pipeline, logging, )`
- `from peft import LoraConfig, PeftModel`
- `from trl import SFTTrainer`

Please review the documentation for each of these libraries on their respective API documents.

Transformers: [https://pypi.org/project/transformers/](https://pypi.org/project/transformers/)
Peft: [https://pypi.org/project/peft/](https://pypi.org/project/peft/)
Trl: [https://pypi.org/project/trl/](https://pypi.org/project/trl/)
```python
• def print_system_specs():
  # Check if CUDA is available
  is_cuda_available = torch.cuda.is_available()
  print("CUDA Available:", is_cuda_available)
  # Get the number of available CUDA devices
  num_cuda_devices = torch.cuda.device_count()
  print("Number of CUDA devices:", num_cuda_devices)
  if is_cuda_available:
    for i in range(num_cuda_devices):
      # Get CUDA device properties
      device = torch.device('cuda', i)
      print(f"--- CUDA Device {i} ---")
      print("Name:", torch.cuda.get_device_name(i))
      print("Compute Capability:", torch.cuda.get_device_capability(i))
      print("Total Memory:", torch.cuda.get_device_properties(i).total_memory, "bytes")
  # Get CPU information
  print("--- CPU Information ---")
  print("Processor:", platform.processor())
  print("System:", platform.system(), platform.release())
  print("Python Version:", platform.python_version())
  print_system_specs()
```

Output:

CUDA Available: True
Number of CUDA devices: 1
--- CUDA Device 0 ---
Name: Tesla T4
Compute Capability: (7, 5)
Total Memory: 15835398144 bytes
--- CPU Information ---
Processor: x86_64
System: Linux 5.15.109+
Python Version: 3.10.12

Free Google Colab offers a 15GB Graphics Card
All the parameters you need

# The model that you want to train from the Hugging Face hub
model_name = "NousResearch/Llama-2-7b-chat-hf"

# The instruction dataset to use
dataset_name = "mlabonne/guanaco-llama2-1k"

# Fine-tuned model name
new_model = "Llama-2-7b-chat-finetune"

# QLoRA parameters

# LoRA attention dimension
lora_r = 64

# Alpha parameter for LoRA scaling
lora_alpha = 16

# Dropout probability for LoRA layers
lora_dropout = 0.1
# bitsandbytes parameters

# Activate 4-bit precision base model loading
use_4bit = True

# Compute dtype for 4-bit base models
bnb_4bit_compute_dtype = "float16"

# Quantization type (fp4 or nf4)
bnb_4bit_quant_type = "nf4"

# Activate nested quantization for 4-bit base models (double quantization)
use_nested_quant = False

# SFT parameters

# Maximum sequence length to use
max_seq_length = None

# Pack multiple short examples in the same input sequence to increase efficiency
packing = False

# Load the entire model on the GPU 0
device_map = {
    "": 0
}
# TrainingArguments parameters

# Output directory where the model predictions and checkpoints will be stored
output_dir = "./results"

# Number of training epochs
num_train_epochs = 1

# Enable fp16/bf16 training (set bf16 to True with an A100)
fp16 = False
bf16 = False

# Batch size per GPU for training
per_device_train_batch_size = 4

# Batch size per GPU for evaluation
per_device_eval_batch_size = 4

# Number of update steps to accumulate the gradients for
gradient_accumulation_steps = 1

# Enable gradient checkpointing
gradient_checkpointing = True

# Maximum gradient normal (gradient clipping)
max_grad_norm = 0.3

# --continue--
# Initial learning rate (AdamW optimizer)
learning_rate = 2e-4

# Weight decay to apply to all layers except bias/LayerNorm weights
weight_decay = 0.001

# Optimizer to use
optim = "paged_adamw_32bit"

# Learning rate schedule
lr_scheduler_type = "cosine"

# Number of training steps (overrides num_train_epochs)
max_steps = -1

# Ratio of steps for a linear warmup (from 0 to learning rate)
warmup_ratio = 0.03

# Group sequences into batches with same length
# Saves memory and speeds up training considerably
group_by_length = True

# Save checkpoint every X updates steps
save_steps = 0

# Log every X updates steps
logging_steps = 25
Training process

Load everything and start the fine-tuning process:

1. load the dataset
2. configuring bitsandbytes for 4-bit quantization
3. loading the Llama 2 model in 4-bit precision on a GPU with the corresponding tokenizer
4. loading configurations for QLoRA, regular training parameters, and passing everything to the SFTTrainer
Training loss and store model

After Training, store our new Llama-2-7b-chat-finetune model:

```python
# Reload model
base_model = AutoModelForCausalLM.from_pretrained(
    model_name,
    low_cpu_mem_usage=True,
    return_dict=True,
    torch_dtype=torch.float16,
    device_map=device_map,
)
model = PeftModel.from_pretrained(base_model, new_model)
model = model.merge_and_unload()

# Reload tokenizer to save it
tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
tokenizer.pad_token = tokenizer.eos_token
tokenizer.padding_side = "right"
```
Push models to HuggingFace

You will use your fine-tuning model for next!

Successful:

Token has not been saved to git credential helper.
Your token has been saved to /root/.cache/huggingface/token
Login successful
CommitInfo(commit_url='https://huggingface.co/MoinFaisal/Llama-2-7b-chat-finetune/commit/329bdec40b9e0f6ec91b8c6f8d9e610fe0e62f1d7', commit_message='Upload tokenizer',
commit_description=' ', old_id='329bdec40b9e0f6ec91b8c6f8d9e610fe0e62f1d7', pr_url='https://huggingface.co/MoinFaisal/Llama-2-7b-chat-finetune/discussions/0', pr_revision='refs/pr/0',
pr_num=0)
Large language models are often used for natural language processing tasks such as text classification, sentiment analysis, and machine translation. They are also used for generating text on a large dataset of text.

Some examples of large language models include:

* BERT (Bidirectional Encoder Representations from Transformers): A popular large language model developed by Google that is trained on a large dataset of text and is designed to gen
What is LangChain?

- a framework for developing applications powered by language models
- connect a language model to sources of context (prompt instructions, few shot examples)
- simplifies the process of creating generative AI application interfaces
Next week

- Provide an in-depth overview of LangChain
- Another part of LLMs application: prompts
- How to build a chat bot
Reference

- https://pypi.org/project/transformers/
- https://pypi.org/project/peft/
- https://pypi.org/project/trl/
- https://www.langchain.com/