SAT 5114

AI in HEALTHCARE PROJECT

Install and load libraries

```
!pip install -qqq "unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git" --progress-bar off
!pip install -qqq xformers trl peft accelerate bitsandbytes triton --progress-bar off
!pip install -qqq unsloth transformers accelerate datasets peft bitsandbytes wandb evaluate bert-score
```

```
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
  Building wheel for unsloth (pyproject.toml) ... done
ERROR: pip's dependency resolver does not currently take into account all the packages that are installed. This behaviour is the source
gcsfs 2025.3.2 requires fsspec==2025.3.2, but you have fsspec 2024.12.0 which is incompatible.
tensorflow-metadata 1.17.1 requires protobuf<6.0.0,>=4.25.2; python_version >= "3.11", but you have protobuf 3.20.3 which is incompatibl
grpcio-status 1.71.0 requires protobuf<6.0dev,>=5.26.1, but you have protobuf 3.20.3 which is incompatible.
ydf 0.11.0 requires protobuf<6.0.0,>=5.29.1, but you have protobuf 3.20.3 which is incompatible.
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.0/84.0 kB 2.5 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.1/61.1 kB 6.1 MB/s eta 0:00:00
```

Import Libraries

```
# Import libraries
import torch
import unsloth
import numpy as np
from tqdm import tqdm
from datasets import load_dataset
from transformers import (
    AutoModelForCausalLM,
    AutoTokenizer,
    TrainingArguments,
    Trainer,
    EarlyStoppingCallback,
    TextStreamer,
    LogitsProcessor,
    LogitsProcessorList,
    BitsAndBytesConfig,
)
from peft import LoraConfig, get_peft_model
from sklearn.metrics import f1_score
import evaluate
```

```
🦥 Unsloth: Will patch your computer to enable 2x faster free finetuning.
Unsloth: Failed to patch Gemma3ForConditionalGeneration.
🦥 Unsloth Zoo will now patch everything to make training faster!
```

Install and define evaluation

```
!pip install rouge_score
bleu_metric = evaluate.load("bleu")
rouge_metric = evaluate.load("rouge")
bertscore_metric = evaluate.load("bertscore")
```

```
Collecting rouge_score
  Downloading rouge_score-0.1.2.tar.gz (17 kB)
  Preparing metadata (setup.py) ... done
Requirement already satisfied: absl-py in /usr/local/lib/python3.11/dist-packages (from rouge_score) (1.4.0)
Requirement already satisfied: nltk in /usr/local/lib/python3.11/dist-packages (from rouge_score) (3.9.1)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from rouge_score) (2.0.2)
Requirement already satisfied: six>=1.14.0 in /usr/local/lib/python3.11/dist-packages (from rouge_score) (1.17.0)
Requirement already satisfied: click in /usr/local/lib/python3.11/dist-packages (from nltk->rouge_score) (8.1.8)
Requirement already satisfied: joblib in /usr/local/lib/python3.11/dist-packages (from nltk->rouge_score) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.11/dist-packages (from nltk->rouge_score) (2024.11.6)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from nltk->rouge_score) (4.67.1)
Building wheels for collected packages: rouge_score
  Building wheel for rouge_score (setup.py) ... done
  Created wheel for rouge_score: filename=rouge_score-0.1.2-py3-none-any.whl size=24934 sha256=8b15b77f151f7057b1fba1daa2b39b6e5a9e9489e
  Stored in directory: /root/.cache/pip/wheels/1e/19/43/8a442dc83660ca25e163e1bd1f89919284ab0d0c1475475148
Successfully built rouge_score
Installing collected packages: rouge_score
Successfully installed rouge_score-0.1.2
```

```
Downloading builder script: 100%  5.94k/5.94k [00:00<00:00, 649kB/s]
Downloading extra modules:  4.07k/? [00:00<00:00, 337kB/s]
Downloading extra modules: 100%  3.34k/3.34k [00:00<00:00, 298kB/s]
Downloading builder script: 100%  6.27k/6.27k [00:00<00:00, 533kB/s]
Downloading builder script: 100%  7.95k/7.95k [00:00<00:00, 818kB/s]
```

Load and prepare the dataset

```
dataset = load_dataset("Shekswess/gemma_medquad_instruct_dataset", split="train[:400]")
dataset = dataset.train_test_split(test_size=0.1)

#lavita/ChatDoctor-HealthCareMagic-100k
#Shekswess/gemma_medquad_instruct_dataset
```

```
README.md: 100%  1.64k/1.64k [00:00<00:00, 197kB/s]
train-00000-of-00001.parquet: 100%  17.9M/17.9M [00:00<00:00, 55.3MB/s]
Generating train split: 100%  16359/16359 [00:00<00:00, 170279.88 examples/s]
```

Format the dataset

```
# Formatting function
def format_instruction(example):
    return {
        "prompt": f"### Instruction:\n{example['instruction']}\n\n### Input:\n{example['input']}\n\n### Response:\n",
        "response": example["output"]
    }

dataset = dataset.map(format_instruction)
```

```
Map: 100%  360/360 [00:00<00:00, 6465.04 examples/s]
Map: 100%  40/40 [00:00<00:00, 2145.51 examples/s]
```

Model setup and quantization

```
# Model setup with 4-bit quantization
model_name = "unsloth/llama-3-8b"
quantization_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16,
    bnb_4bit_use_double_quant=True,
)

tokenizer = AutoTokenizer.from_pretrained(model_name)
tokenizer.pad_token = tokenizer.eos_token
```

```
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    quantization_config=quantization_config,
    device_map="auto",
    use_cache=False,
)
```

```
Loading checkpoint shards: 100%  4/4 [00:18<00:00,  3.82s/it]
```

LoRA configuration and PEFT

```
# Improved LoRA configuration
peft_config = LoraConfig(
    r=4,
    lora_alpha=8,
    target_modules=["q_proj", "v_proj", "k_proj", "o_proj"],
    lora_dropout=0.05,
    bias="lora_only",
    task_type="CAUSAL_LM",
    inference_mode=False,
)
model = get_peft_model(model, peft_config)
model.enable_input_require_grads()
```

Data Collation and Tokenization

```
# Add data collator
from transformers import DataCollatorForSeq2Seq

data_collator = DataCollatorForSeq2Seq(
    tokenizer,
    pad_to_multiple_of=8,
    padding=True,
    return_tensors="pt",
)

# Update tokenize function
def tokenize_function(examples):
    texts = [p + r for p, r in zip(examples["prompt"], examples["response"])]
    tokenized = tokenizer(
        texts,
        max_length=512,
        truncation=True,
        padding="max_length",
        add_special_tokens=False
    )
    return {
        "input_ids": tokenized["input_ids"],
        "attention_mask": tokenized["attention_mask"],
        "labels": tokenized["input_ids"].copy()
    }

tokenized_dataset = dataset.map(tokenize_function, batched=True)
```

```
Map: 100%  360/360 [00:00<00:00, 2158.53 examples/s]
Map: 100%  40/40 [00:00<00:00, 1231.32 examples/s]
```

Metrics calculation

```
# Metrics calculation (updated)
def compute_metrics(eval_pred):
    preds, labels = eval_pred
    labels = np.where(labels != -100, labels, tokenizer.pad_token_id)

    # Convert logits to token IDs (shape: [batch_size, seq_length])
    preds = np.argmax(preds, axis=-1)  # Add this line
```

```
    pred_texts = tokenizer.batch_decode(preds, skip_special_tokens=True)
    label_texts = tokenizer.batch_decode(labels, skip_special_tokens=True)

    results = {}

    # BLEU
    results["bleu"] = bleu_metric.compute(
        predictions=pred_texts,
        references=[[text] for text in label_texts]
    )["bleu"]

    # ROUGE
    results.update(rouge_metric.compute(
        predictions=pred_texts,
        references=label_texts
    ))

    # BERTScore
    bert_results = bertscore_metric.compute(
        predictions=pred_texts,
        references=label_texts,
        lang="en"
    )
    results["bert_score"] = np.mean(bert_results["f1"])

    return results
```

Setup Training arguments and trainer

```
# Modified TrainingArguments with evaluation strategy
training_args = TrainingArguments(
    output_dir="./llama3_healthcare",
    per_device_train_batch_size=2,
    per_device_eval_batch_size=4,
    gradient_accumulation_steps=1,
    save_strategy ='epoch',      # Changed to "epoch" to enable saving
    logging_strategy="no",     # Disable logging
    learning_rate=1e-5,
    weight_decay=1,
    num_train_epochs=4,
    lr_scheduler_type="cosine",
    warmup_ratio=0.1,
    logging_steps=50,
    load_best_model_at_end=True,
    fp16=True,
    gradient_checkpointing=True,
    report_to="none",        # Disabled all reporting
    remove_unused_columns=False,
    # Add evaluation strategy for EarlyStoppingCallback
    eval_strategy = "epoch"  # or "steps" with logging_steps defined
)

# Trainer remains the same
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["test"],
    compute_metrics=compute_metrics,
    callbacks=[EarlyStoppingCallback(early_stopping_patience=3)]
)

# Training will now only log to console
trainer.train()
```

[720/720 06:31, Epoch 4/4]

Epoch	Training Loss	Validation Loss	Bleu	Rouge1	Rouge2	Rougel	Rougelsum	Bert Score
1	No log	0.809389	0.377964	0.685381	0.374715	0.575169	0.605812	0.883580
2	No log	0.665850	0.452822	0.738861	0.455310	0.651004	0.665704	0.905270
3	No log	0.642137	0.459082	0.738626	0.456576	0.652674	0.665940	0.905385
4	No log	0.639166	0.459274	0.739225	0.457579	0.652887	0.666850	0.905519

```
tokenizer_config.json: 100%  25.0/25.0 [00:00<00:00, 2.99kB/s]
config.json: 100%  482/482 [00:00<00:00, 61.0kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 16.8MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 12.4MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 19.0MB/s]
Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to regular HTTP download. For better perfc
WARNING:huggingface_hub.file_download:Xet Storage is enabled for this repo, but the 'hf_xet' package is not installed. Falling back to r
model.safetensors: 100%  1.42G/1.42G [00:02<00:00, 1.34GB/s]
Some weights of RobertaModel were not initialized from the model checkpoint at roberta-large and are newly initialized: ['pooler.dense.b
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
TrainOutput(global_step=720, training_loss=1.4614861382378472, metrics={'train_runtime': 392.4554, 'train_samples_per_second': 3.669,
'train_steps_per_second': 1.835, 'total_flos': 3.321446050824192e+16, 'train_loss': 1.4614861382378472, 'epoch': 4.0})
```

Model saving

```
# Save and generation code remains unchanged
model.save_pretrained("./llama3_healthcare_finetuned")
tokenizer.save_pretrained("./llama3_healthcare_finetuned")
```

```
('./llama3_healthcare_finetuned/tokenizer_config.json',
 './llama3_healthcare_finetuned/special_tokens_map.json',
 './llama3_healthcare_finetuned/tokenizer.json')
```

Text generation

```
from transformers import pipeline # Import the pipeline function
generator = pipeline(
    "text-generation",
    model=model,
    tokenizer=tokenizer,

    max_new_tokens=256,
    do_sample=True,
    temperature=0.7,
    top_p=0.9,
    repetition_penalty=1.2,
)

test_question = "What is the management for hypertension?"
result = generator(test_question, num_return_sequences=1)
print("\nGenerated Response:")
print(result[0]['generated_text'])
```

```
Device set to use cuda:0
The model 'PeftModelForCausalLM' is not supported for text-generation. Supported models are ['AriaTextForCausalLM', 'BambaForCausalLM',

Generated Response:
What is the management for hypertension? How to control it?
Hypertension can be controlled by a combination of lifestyle changes and medications. It’s important that you make these modifications b
Lifestyle Changes - The first line treatment should always include dietary modification (i.e., reduction in salt intake), physical activ
Medications - If blood pressure remains uncontrolled after three months despite following all advice above then doctors usually recommen
A variety of different medicines exist which treat high BP including diuretics (water pills); beta-blockers like Atenolol or Metoprolol;
```

Start coding or generate with AI.

Add Graphical User Interface for input and output texts

Install GUI library gradio

```
!pip install gradio
```

Show hidden output

Define predicition Function

```
import gradio as gr

def predict(input_text):
  result = generator(input_text, num_return_sequences=1)
  return result[0]['generated_text']
```

Create the user interface

```
iface = gr.Interface(
    fn=predict,
    inputs=gr.Textbox(lines=2, placeholder="Enter your question here..."),
    outputs="text",
    title="Medical Question Answering",
    description="Ask questions about medical topics and get answers from our AI model."
)
```

```
iface.launch()
```

```
Rerunning server... use `close()` to stop if you need to change `launch()` parameters.
----
It looks like you are running Gradio on a hosted a Jupyter notebook. For the Gradio app to work, sharing must be enabled. Automatically

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://ecefae7ece90590802.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working dir
```