**Optimizing Large Language Model Training with Hugging Face AutoTrain Advanced**

In this project we unlock the potential of large language models (LLMs) with AutoTrain-Advanced, a powerful no-code solution that allows us to train ML models. In this guide, we'll walk you through the steps to fine-tune the TinyPixel/Llama-2-7B-bf16-sharded model using cutting-edge techniques and tools.

**Introduction**

As the demand for sophisticated language models continues to rise, so does the need for efficient training methodologies. AutoTrain-Advanced emerges as a solution, offering advanced capabilities to overcome challenges such as memory limitations and compute resources.

Addressing Training Challenges: Traditionally, training large language models posed significant challenges, especially when dealing with limited resources. AutoTrain-Advanced addresses these challenges by optimizing parameters like quantization, learning rates, and batch sizes, ensuring efficient utilization of available hardware.

**Getting started with AutoTrain-Advanced:**

We'll demonstrate how to fine-tune the TinyPixel/Llama-2-7B-bf16-sharded model using AutoTrain-Advanced. By leveraging its capabilities, we can achieve optimal performance even with constrained resources. From setting up the environment to configuring training parameters.

We start by installing and upgrading the autotrain-advanced package in Colab environment. For this to work properly, we need a python version of 3.8 or greater and access to GPU. Then we install the latest version of huggingface\_hub. This library provides tools for interacting with Hugging Face hub, such as downloading models, creating repositories, or managing your models programmatically.

!pip install -U autotrain-advanced

!pip install -U huggingface\_hub

To run autotrain-advanced on Colab, we also need to run the code below:

!autotrain setup --update-torch

Then provide your Hugging Face token to successfully log in to Hugging Face:

from huggingface\_hub import notebook\_login

notebook\_login()

**Dataset Generation:**

We used ChatGPT to create a dataset of random possible issues with Apple products and services. Then we asked ChatGPT to write a description for each issue that includes a detailed description of the issue, the product’s model, date purchased, their location, and their internet provider. The third column must have a specific format for Llama to be trained. For starters, it must be labeled ‘text’ and include Human/Assistant pair. We used the following code to generate this column:

df['text'] = df.apply(lambda row: f"###Human:\ngenerate a detailed description prompt for {row['Issue']}\n\n###Assistant:\n{row['Description']}", axis=1)

Our dataset includes 325 examples. A snapshot showing part of the dataset is provided below:

A group of lines on a white background

Description automatically generated

**Optimizing Training Parameters:**

AutoTrain-Advanced offers multiple options to fine-tune training parameters. From adjusting learning rates to optimizing batch sizes and epochs, every aspect can be calibrated to maximize performance while minimizing resource usage.

!autotrain llm --train\

--project-name 'Apple-Issue-Prompt'\

--model TinyPixel/Llama-2-7B-bf16-sharded\

--data-path .\

--text-column text\

--use-peft\

--quantization int4 \

--lr 2e-4\

--train-batch-size 4\

--epochs 8\

--trainer sft > training.log &

We started by training the model on 240 samples of data. The trained model was not able to generate acceptable and meaningful responses. When we increased the data to 325 samples, we were able to see consistent logical responses. We also tried different values for learning rate, batch size, and epoch. What seemed to be more effective in reducing the loss was increased epoch. We iteratively increased epoch and measured the loss. At 8 epochs, loss was around 0.5 and at 20 epochs it was reduced to 0.29. We decided to continue with epoch = 8 for two reasons; first, increasing the number of epochs didn’t seem to change the generated response and second, we could run the risk of overfitting on the training data. When a larger dataset is available, it is advised to set aside some validation data to measure the fine-tuned model’s loss and generalizability.

import subprocess

# Define range of epochs to iterate through

min\_epochs = 8

max\_epochs = 21

# Dictionary to store training loss for each epoch

training\_loss = {}

# Iterate through epochs

for epoch in range(min\_epochs, max\_epochs , 5):

    # Construct command with current number of epochs

    !autotrain llm --train\

    --project-name 'Apple-Issue-Prompt'\

    --model TinyPixel/Llama-2-7B-bf16-sharded\

    --data-path .\

    --text-column text\

    --use-peft\

    --quantization int4 \

    --lr 2e-4\

    --train-batch-size 4\

    --epochs {epoch}\

    --trainer sft > training.log &

    training\_loss[epoch] = extract\_train\_loss('training.log')

print(f"Optimal number of epochs: {training\_loss[0]}")

**Real-World Application:**

To showcase the practical implications of AutoTrain-Advanced, we'll generate detailed descriptions for common tech support issues using the fine-tuned model. From troubleshooting Apple Watch screen blinking to iCloud syncing issues on MacBook, the model exhibits good proficiency in addressing user queries.

import peft

import torch

from transformers import AutoModelForCausalLM, AutoTokenizer

from torch.nn import DataParallel

tokenizer = AutoTokenizer.from\_pretrained("/content/Apple-Issue-Prompt")

model = AutoModelForCausalLM.from\_pretrained("/content/Apple-Issue-Prompt")

input\_context = '''

###Human:

Generate a detailed description prompt for apple watch screen blinking.

###Assistant:

'''

input\_ids = tokenizer.encode(input\_context, return\_tensors="pt")

output = model.generate(input\_ids, min\_length = 80, max\_length=120, do\_sample=True, temperature=0.3, num\_return\_sequences=1)

generated\_text = tokenizer.decode(output[0], skip\_special\_tokens=True)

print(generated\_text)

oading checkpoint shards: 100%

 14/14 [00:08<00:00,  1.59it/s]

###Human:

Generate a detailed description prompt for apple watch screen blinking.

###Assistant:

Customer's Apple Watch Series 5 purchased in November 2021 is experiencing the screen blinking randomly. They reside in Richmond and have Xfinity as their internet provider. They have tried restarting the watch and updating to the latest software, but the issue persists.

input\_context = '''

###Human:

Generate a detailed description prompt for icloud on macbook is not syncing.

###Assistant:

'''

input\_ids = tokenizer.encode(input\_context, return\_tensors="pt")

output = model.generate(input\_ids, min\_length = 80, max\_length=120, do\_sample=True, temperature=0.3, num\_return\_sequences=1)

generated\_text = tokenizer.decode(output[0], skip\_special\_tokens=True)

print(generated\_text)

###Human:

Generate a detailed description prompt for icloud on macbook is not syncing.

###Assistant:

Customer's iCloud on their MacBook Pro 13-inch purchased in July 2021 is not syncing. They are located in Honolulu and have Hawaiian Telcom as their internet provider. They have tried signing out and back into iCloud, but the issue persists.

**Conclusion:**

In this work, AutoTrain-Advanced was used to fine-tune Llama-2-7B-bf16-sharded model on a dataset comprised of Apple’s issue description pairs. This project provided an example of fine-tuning the Llama 2 model using techniques like Hugging Face libraries like auto-train advanced, PEFT, and transformers to overcome limited memory and compute resources.