OTIMIZAÇÃO DE MODELOS LLM PARA GERAÇÃO DE RESPOSTAS UTILIZANDO FINE-TUNING

Cintia Zago, Guilherme Machado, Rodrigo Vidal
cintiazago@gmail.com / 26guilhermemachado@gmail.com / rodrigo.vidal@fbsbrasil.com.br
FIAP – Faculdade de Informática e Administração Paulista
São Paulo, Brasil

***Resumo* — este documento apresenta a implementação de um *foundational model* (LLaMA) e a execução de fine-tuning utilizando um dataset existente que receberá perguntas com um contexto obtido deste arquivo e, a partir de um prompt formado pela pergunta do usuário sobre o título do produto, o modelo deverá gerar uma resposta baseada na pergunta do usuário trazendo, como resultado do aprendizado do fine-tuning, os dados da sua descrição.**

Keywords— python;large language model; fine tuning; inteligência artificial;

I. Introdução

Os *foundational models* são modelos de inteligência artificial de larga escala pré-treinados que servem como uma base para diversas tarefas específicas. Treinados em extensos conjuntos de dados, esses modelos aprenderam representações de dados de uso geral, tornando-os mais adequados para fine-tuning em aplicações específicas, como classificação de texto, reconhecimento de imagens e tradução de idiomas. Exemplos destes modelos incluem BERT, GPT-3 e LLaMA, que podem ser adaptados para uma ampla gama de casos de uso com um treinamento adicional com baixo esforço. Assim, este projeto tem como objetivo realizar o fine-tuning - que é o processo de se otimizar o modelo em vez de treiná-lo do zero, o que pode ser muito custoso em termos de tempo e recursos computacionais - do modelo pré-treinado utilizando o *foundational model* LLaMA.

II. Descrição do Problema

O problema abordado neste artigo consiste em utilizar o dataset *"The AmazonTitles-1.3MM"*, que contém consultas textuais reais de usuários e títulos associados à produtos relevantes encontrados na Amazon e suas respectivas descrições, medidos por ações implícitas ou explícitas dos usuários; para treinar um *foundational model* - neste caso, utilizaremos o modelo LLaMA, criado pela Meta AI - e executar o fine-tuning do mesmo. Este modelo será capaz de receber perguntas por comandos em um contexto contido neste dataset. Assim, a partir do prompt de comando formado pela pergunta do usuário sobre o título do produto, este modelo deve gerar uma resposta com base nesta pergunta como resultado do aprendizado do fine-tuning.

O maior desafio deste problema encontra-se em lidar com o fine-tuning de um dataset que contém milhões de registros. Devido a isto, o pré-processamento deste faz-se necessário para reduzirmos o tamanho do arquivo e termos dados relevantes para o treinamento.

III. Pré-Processamento

Antes de iniciarmos o fine-tuning do modelo escolhido, é necessário realizar o pré-processamento deste arquivo para que os dados estejam concisos e contenham apenas os registros relevantes.

Primeiramente, o arquivo foi convertido para o formato *.csv* para melhor leitura e visualização dos dados. Assim, conseguimos também utilizar apenas as colunas que precisamos para treinar o modelo, que neste caso, são a coluna *title* e a coluna *content.* Foi utilizada a biblioteca *polar* para manipular o arquivo, pois esta apresenta uma melhor performance com arquivos muito grandes se comparada com a biblioteca *pandas*. Após esta etapa, realizamos a remoção dos registros nulos e vazios, gerando uma redução no tamanho do arquivo de mais de 858 mil linhas, saindo de mais de 2 milhões de linhas para 1,39 milhões, aproximadamente. Ao final do processamento, exportamos o arquivo gerado para o formato *parquet,* que, segundo um estudo realizado pelo *Databricks*[1], pode gerar uma economia de recursos de armazenamento de até 87% e uma redução significativa no tempo de execução de consultas quando utilizamos este formato.

IV. Execução do Fine-Tuning

Fine-tuning é uma técnica utilizada no campo da Inteligência Artificial (IA), particularmente em modelos de aprendizado de máquina e aprendizado profundo, para adaptar um modelo pré-treinado a uma tarefa específica ou a um conjunto de dados particular. Essa abordagem pode ser extremamente benéfica para empresas que buscam construir soluções personalizadas para suas necessidades específicas.

O ponto de partida do fine tuning é um modelo de IA que já foi treinado em um grande conjunto de dados geral, conhecido como modelo pré-treinado e que neste artigo utilizará o modelo LLaMA treinado com um dataset preparado com mais de um milhão de registros.

O processo de fine-tuning consiste em continuar o treinamento do modelo pré-treinado em um conjunto de dados específico. Esta etapa ajusta os pesos do modelo para que ele se torne mais eficaz para a tarefa específica de interesse.

Durante o fine-tuning, os hiperparâmetros do modelo, como taxa de aprendizado e número de épocas de treinamento, são frequentemente ajustados para otimizar o desempenho do modelo na nova tarefa. A escolha de hiperparâmetros não é

trivial e nem única. A constante avaliação do modelo deve ajudar a escolher novos valores.

Por fim, é importante avaliar o desempenho do modelo ajustado e realizar eventuais refinamentos para melhorar a precisão e a eficácia do modelo na tarefa desejada.

A seguir, na **Figura 1**, podemos ver uma pipeline ilustrando o processo de fine-tuning.

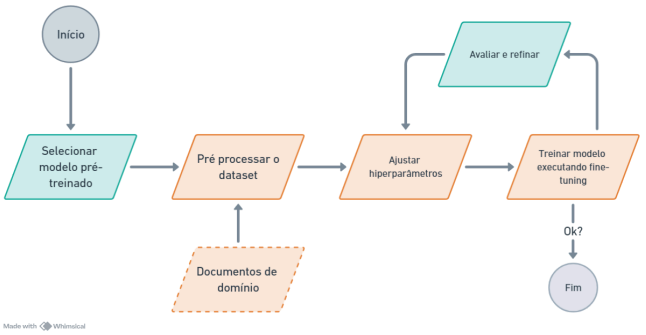

Figura 1: Pipeline sugerida para fine-tunig

V. Resultados e discussão

A seguir serão descritos a parametrização e os resultados obtidos pelo modelo antes e após o treinamento executando fine-tuning:

1. *Resultados do modelo pré-treinamento*
   Para o treinamento do modelo foi utilizado o *foundational model* LLaMa 3.1, e as bibliotecas *unsloth* e *trl.* Na **Figura 2** pode-se observar os parâmetros utilizados para o carregamento do modelo.

```
"""
Define some model constants.
"""

import json
import torch

from datasets import load_dataset
from transformers import TrainingArguments
from trl import SFTTrainer
from unsloth import FastLanguageModel, is_bfloat16_supported

max_seq_length = 2048
dtype = None
load_in_4bit = True
fourbit_models = [
    "unsloth/mistral-7b-v0.3-bnb-4bit",
    "unsloth/mistral-7b-instruct-v0.3-bnb-4bit",
    "unsloth/llama-3-8b-bnb-4bit",
    "unsloth/llama-3-8b-Instruct-bnb-4bit",
    "unsloth/llama-3-70b-bnb-4bit",
    "unsloth/Phi-3-mini-4k-instruct",
    "unsloth/Phi-3-medium-4k-instruct",
    "unsloth/mistral-7b-bnb-4bit",
    "unsloth/gemma-7b-bnb-4bit",
]
```

Figura 2: Carregamento do modelo pré-treinamento

Na definição dos parâmetros, foram utilizados:

A. *max_seq_length:* para definir o número máximo de tokens.
B. *dtype:* define os pesos do nosso modelo, e quando é configurado como None, nos permite utilizar o valor padrão.
C. *load_in_bit:* que permite carregar o modelo com precisão de 4 bit, o que vai proporcionar uma performance melhor, porém pode-se perder um pouco de precisão no resultado.

Ao realizarmos os testes enviando prompts com perguntas para este modelo, nota-se que a resposta obtida se perde um pouco, gerando um conteúdo mais extenso, e que não responde um dos questionamentos feitos a respeito do valor do produto.

2. *Resultados do modelo pós otimização com fine-tuning*
   Para treinar o modelo executando fine-tuning, utilizamos as bibliotecas *unsloth, trl* e o método *TrainingArguments* da biblioteca *transformers* para configurar o treinamento.
   Na **Figura 3**, são apresentados os parâmetros que foram necessários para obtermos o resultado esperado.

```
from trl import SFTTrainer
from transformers import TrainingArguments
from unsloth import is_bfloat16_supported

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = train_ds,
    dataset_text_field = "prompt",
    max_seq_length = max_seq_length,
    dataset_num_proc = 2,
    packing = False, # Can make training 5x faster for short sequences.
    args = TrainingArguments(
        per_device_train_batch_size = 8,
        gradient_accumulation_steps = 4,
        warmup_steps = 10,
        num_train_epochs = 1, # Set this for 1 full training run.
        max_steps = 60,
        learning_rate = 2e-4,
        fp16 = not is_bfloat16_supported(),
        bf16 = is_bfloat16_supported(),
        logging_steps = 1,
        optim = "adamw_8bit",
        weight_decay = 0.01,
        lr_scheduler_type = "linear",
        seed = 3407,
        output_dir = "outputs",
    ),
)
```

Figura 3: Parametrização do treinamento do modelo com fine-tuning

Na definição dos parâmetros, alguns utilizados foram:

A. *per_device_train_batch_size*: Tamanho do batch por dispositivo (GPU ou CPU). Define o número de exemplos processados por cada dispositivo antes de atualizar os pesos do modelo.

B. *gradient_accumulation_steps*: Número de *steps* para acumular gradientes antes de atualizar os pesos do modelo. Esse valor permite simular batches maiores mesmo quando há limitação de memória.
C. *warmup_steps*: Número de passos de warmup (aquecimento) para aumentar a taxa de aprendizado de forma gradual no início do treinamento. Ajuda a estabilizar o processo de ajuste do modelo.
D. *num_train_epochs*: Número de épocas de treinamento. O valor 1 indica que o dataset será percorrido completamente uma vez.
E. *max_steps*: Número máximo de steps (passos) para o treinamento. Limita o número de iterações durante o treinamento. Quando especificado, o valor de num_train_epochs é ignorado.
F. *learning_rate*: Taxa de aprendizado usada pelo otimizador. Controla a velocidade com que o modelo ajusta seus pesos.
G. *logging_steps*: Intervalo de steps para registrar métricas de desempenho e progresso do treinamento. O valor 1 indica que o progresso será registrado a cada passo.
H. *optim*: Tipo de otimizador a ser utilizado. O valor "adamw_8bit" indica a utilização do AdamW com precisão reduzida (8 bits), o que economiza memória e pode acelerar o treinamento.
I. *weight_decay*: Fator de decaimento de peso para regularização. Reduz o valor dos pesos do modelo ao longo do tempo para evitar *overfitting*.
J. *lr_scheduler_type*: Tipo de agendador de taxa de aprendizado. O valor "linear" indica que a taxa de aprendizado será decrescida linearmente ao longo do tempo.
K. *seed*: Semente utilizada para inicializar os geradores aleatórios, garantindo a reprodutibilidade dos resultados.

Estes parâmetros controlam diferentes aspectos do treinamento, como o gerenciamento do dataset, configuração de hardware, e ajustes do otimizador. Ao alterarmos estes ajustes podemos obter um resultado melhor ou pior, por isto é importante a avaliação das respostas após cada treinamento.

VI. Conclusão

Após ser executada a otimização do modelo com fine-tuning foi possível perceber uma considerável melhora nas respostas geradas.

No modelo pré-treinado utilizando o *foundational model* LLaMA, percebe-se que ao adicionar algumas questões, ele se perde um pouco e não responde, por exemplo, ao questionamento a respeito do preço do produto, além do mais, a resposta se estende com muitas informações.

Já no modelo pós treinamento, pode-se notar que, além da resposta ser mais sucinta, ele consegue identificar que o preço do produto depende de outras variáveis e que, portanto, não pode definir um preço sem que no prompt adicionemos estas informações mais precisas. No entanto, foi notável que as respostas em ambos os modelos poderiam ser melhores e mais precisas caso o *dataset* tivesse uma qualidade superior de dados para o treinamento dos modelos.

Conclui-se que a execução do fine-tuning pode aumentar a precisão consideravelmente do resultado do modelo sem muito esforço tanto em relação à implementação, quanto aos recursos necessários para esta otimização.

Referências

[1] Databricks disponível em: https://www.databricks.com/glossary/what-is-parquet. Acessado em 29 de setembro de 2024.