Tech Challenge 3 – Gustavo Molina Figueiredo RM 359124

Fine-tuning de um modelo pré-treinado

O primeiro passo a ser realizado consiste em configurar o ambiente para *fine-tuning* de modelos de linguagem em GPUs com memória limitada. Foram instaladas bibliotecas como PyTorch (com suporte a CUDA), Unsloth (para otimização e quantização em 4 bits), e ferramentas do ecossistema Hugging Face, como Transformers, Datasets e peft. Isso permite o ajuste eficiente de LLMs, reduzindo o uso de memória e acelerando o treinamento.

A seguir, são realizados o carregamento e a configuração do modelo pré-treinado Llama-3-8b utilizando a biblioteca unsloth com suporte para quantização em 4 bits.

Inicialmente, verifica-se se um modelo já ajustado (por fine-tuning) está salvo no Google Drive. Caso haja um modelo, ele é carregado junto com o tokenizador para reutilização.

Se não houver um modelo salvo, o código faz o download do modelo base (foundation model), configura diversos hiperparâmetros, como a utilização de float16 para otimizar o consumo de memória.

Após o carregamento do modelo base, são enviadas 15 perguntas sobre os produtos contidos no dataset, para que a acurácia das respostas após o fine-tuning possa ser comparada.

Em seguida, aplica-se o método LoRA (Low-Rank Adaptation), que é uma técnica desenvolvida para reduzir custos computacionais e de memória no processo de fine-tuning de LLMs, como o LLaMA ou GPT. O gradient checkpointing também é ativado para economizar memória durante o treinamento.

Por fim, é realizado um teste de inferência, utilizando um exemplo simples para validar se o modelo está pronto para ser utilizado no processo de fine-tuning.

A seguir é realizada a preparação do dataset para o fine-tuning.

Geração do Dataset

Caso o dataset final não exista, o código utiliza a API GPT-3.5 Turbo da OpenAI para gerar exemplos de perguntas e respostas baseadas em informações dos produtos contidas no arquivo trn.json.

Cada exemplo segue o padrão: "*Context: [Produto] Instruction: [Pergunta] Response: [Resposta]*"

Os dados processados são armazenados em um novo arquivo JSON.

Conversão para Dataset Hugging Face

Os dados são transformados em um formato compatível com o Hugging Face Dataset, pronto para ser utilizado no treinamento.

Agora, o fine-tuning é configurado e executado, utilizando o framework Hugging Face transformers e a classe SFTTrainer da biblioteca trl.

Configuração Inicial

- Desativação do Weight and Biases (W&B), uma ferramenta de rastreamento de experimentos, pois exigia um token para acesso.
- Atualização das bibliotecas Torch e TorchAO para garantir a compatibilidade com configuração de hardware e otimizações mais recentes.

Passando o modelo e o tokenizer preparados para o SFTTrainer

Nesta etapa, são definidos alguns parâmetros:

- train_dataset: O dataset preparado anteriormente.
- dataset_text_field: Campo usado como entrada de texto para o modelo ('text').
- max_seq_length: Define o comprimento máximo das sequências de entrada (256 tokens).
- dataset_num_proc: Número de subprocessos para o processamento paralelo do dataset.
- packing: Desativado (False) para evitar empacotamento adicional de dados.
- per_device_train_batch_size: Lotes de tamanho 3 para cada dispositivo.
- gradient_accumulation_steps: Gradientes acumulados a cada 8 passos, efetivamente aumentando o tamanho do lote.
- max_steps: O treinamento é limitado a 250 passos, adequado para experimentação rápida.
- Otimizador: AdamW com optim="adamw_hf".
- Decaimento de peso (weight_decay): 0.01, para regularização.
- Esquema de ajuste de taxa de aprendizado: cosine (uma redução suave e contínua).
- Taxa de aprendizado inicial: 5e-5.
- Gradientes em ponto flutuante de 16 bits (fp16=True) para economizar memória.
- Gradient checkpointing: Ativado para reduzir o uso de memória durante o backpropagation.
- logging_steps: Logs gerados a cada passo.
- warmup_ratio: 5% dos passos usados para warm-up.
- output_dir: Os resultados são salvos no diretório outputs da sessão.

Execução do Treinamento

O treinamento é iniciado, executando as etapas de backpropagation e ajuste de pesos do modelo com base no dataset fornecido.

O modelo ajustado e seu tokenizador são salvos em um diretório do Google Drive, após o fine-tuning.

Isso é importante para que o modelo e o tokenizador, possam ser reutilizados sem necessidade de realizar o processo de fine-tuning novamente. Além disso, desta forma, os resultados do treinamento são armazenados de forma segura para serem acessados futuramente, mesmo após a finalização da sessão no Colab.

A seguir, as mesmas 15 perguntas enviadas antes ao modelo são enviadas novamente para verificar como o modelo responde após o fine-tuning.

Os prompts solicitam ao modelo responder a perguntas específicas sobre as informações de produtos, como detalhes sobre tamanho, cor, autor, etc.

Após receber a resposta para cada prompt, usa-se regex para extrair a parte mais relevante da resposta gerada.

Conclusão

Observou-se que, apesar do número razoável de 250 iterações, o modelo não convergiu, ou seja, não se notou uma redução significativa do loss. Além disso, o modelo ajustado ainda comete alguns equívocos ao responder algumas perguntas feitas sobre os produtos contidos no dataset inicial.

Algumas das razões pelas quais isso pode estar acontecendo, bem como possíveis soluções, incluem:

- **Qualidade do Dataset**: O dataset talvez não esteja na melhor estrutura, o que pode resultar em respostas imprecisas. Além disso, o tamanho do dataset talvez seja insuficiente para o modelo generalizar adequadamente. Aumentar o dataset, incluindo mais exemplos ou realizando uma limpeza de dados mais rigorosa, poderia melhorar a precisão das respostas.
- **Limitações do Fine-Tuning**: Talvez seja necessário aperfeiçoar a combinação de valores de número de passos de treinamento, taxa de aprendizado e parâmetros do otimizador, visando a melhoria do fine-tuning. Ajustar os hiperparâmetros de forma mais cuidadosa pode ajudar a encontrar a configuração ideal para o treinamento.
- **Escolha do Foundation Model**: Se o modelo inicial escolhido para o fine-tuning não é adequado para o tipo específico de tarefa (no caso, resposta a perguntas baseadas em descrições de produtos), o modelo pode não ser capaz de generalizar bem para o tipo de dado fornecido. Talvez utilizar modelos especializados neste tipo de tarefa, como T5 ou BERT, que são adaptados para questions-answers, pode trazer melhorias.
- **Uso inadequado de técnicas de regularização**: Técnicas como gradient_checkpointing e lora, podem ser eficazes em reduzir o uso de memória e melhorar a eficiência do treinamento, mas, por outro lado, podem impactar negativamente a qualidade do modelo. Uma boa ideia neste caso seria utilizar outras técnicas de otimização mais sofisticadas.