Capítulo 23 – Como modificar um Modelo Gigante (Fine-Tuning e LoRA)

Aléssio Miranda Jr.

Outubro - 2026/2

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

O que você verá neste capítulo

A crença de que Grandes Modelos de Linguagem (LLMs) são monopólio de corporações multibilionárias caiu por terra. Neste capítulo, iniciaremos nossa jornada no **Arco 4: Agentes Autônomos e Módulos Avançados**. O foco desta aula é o *Fine-Tuning* (Ajuste Fino) — a arte de modificar o comportamento de modelos gigantescos em hardware modesto. Introduziremos formalmente a técnica PEFT (Parameter-Efficient Fine-Tuning) e dissecaremos a matemática genial por trás do algoritmo **LoRA** (Low-Rank Adaptation). Por fim, apresentaremos o ecossistema *Unsloth* e o *Hugging Face*, capacitando você a forjar seu próprio "Especialista" em uma única placa de vídeo.

1 Introdução: O Mito do Treinamento Exclusivo

Por muito tempo, consolidou-se a crença de que a criação e modificação de Grandes Modelos de Linguagem (LLMs) era um privilégio exclusivo de corporações trilionárias, como OpenAI, Google ou Meta. O argumento principal era o custo computacional absurdo: treinar um modelo do zero (o chamado *Pre-training*) com bilhões de parâmetros exige a orquestração de milhares de GPUs rodando ininterruptamente por meses. Tal empreitada consome milhões de dólares apenas em energia e resfriamento de infraestrutura.

No entanto, a engenharia de software sempre encontra caminhos para a eficiência. Há uma diferença brutal e fundamental entre **criar** um modelo do zero e **adaptá-lo** para uma nova tarefa. A esmagadora maioria das aplicações

corporativas modernas não precisa de uma IA que tenha lido a internet inteira novamente a cada nova demanda. O que precisamos é pegar um modelo já inteligente e generalista (como o Llama-3 de 8 bilhões de parâmetros) e injetar nele conhecimentos específicos de um domínio, ou alterar radicalmente sua forma de se comunicar. Esse processo é chamado de *Fine-Tuning* (Ajuste Fino).

• Teoria: title=A Diferença entre RAG e Fine-Tuning

Uma confusão clássica no mercado corporativo é quando usar RAG e quando usar Fine-Tuning.

- **RAG (Retrieval-Augmented Generation):** Usado para fornecer *conhecimento factual e mutável*. Se o manual da empresa atualiza toda semana, o RAG é a solução, pois ele busca nos documentos. O modelo aprende "o quê".
- **Fine-Tuning:** Usado para modificar o *comportamento e o formato*. Se você precisa que o modelo responda **sempre** em um formato JSON estrito, ou que adote a voz de um médico neurologista, o Fine-Tuning internaliza o comportamento. O modelo aprende "como".

Tabela 1: Comparativo de estratégias de adaptação de LLMs

Característica	RAG	Full Fine-Tuning	PEFT / LoRA
Objetivo Principal	Fatos atualizados e contexto dinâmico.	Injetar conhecimento profundo.	Modificar formato, tom e comportamento.
Custo Computacional	Baixíssimo (busca vetorial leve).	Altíssimo (clusters de GPUs).	Baixo (uma única GPU gamer/Colab).
Tamanho Gerado	Depende do Banco de Dados.	Gigantesco (cópia total de 16GB+).	Minúsculo (apenas *adapters*, ~50MB).

2 PEFT e a Mágica do LoRA

Se o Fine-Tuning clássico (Full Fine-Tuning) exige atualizar todos os bilhões de pesos de uma rede neural (o que ainda requereria múltiplas placas de vídeo poderosas), como estudantes e startups conseguem fazer isso hoje em

computadores convencionais? A resposta está na sigla **PEFT** (*Parameter-Efficient Fine-Tuning*).

A premissa do PEFT é elegantemente simples: em vez de recalcular e atualizar todos os pesos de uma rede gigantesca durante o treinamento, mantemos a grande maioria (*backbone*) congelada e treinamos apenas um subconjunto minúsculo de novos parâmetros adicionais.

A técnica mais famosa sob o guarda-chuva do PEFT, responsável pela atual revolução Open Source, é o **LoRA** (*Low-Rank Adaptation*), proposta por pesquisadores da Microsoft (Hu et al., 2021).

▶ Prática: title=A Matemática do LoRA em Termos Simples

Para entender o LoRA matematicamente sem entrar em detalhes complexos de álgebra linear avançada, imagine a matriz de pesos de uma camada do modelo original, que possui dimensões gigantescas. Digamos, 10.000×10.000 (resultando em 100 milhões de parâmetros).
O algoritmo LoRA **congela** essa matriz principal. Para aprender os "novos" conhecimentos, ele cria duas matrizes muito menores lado a lado, chamadas de matrizes de adaptação A e B.

- Uma matriz A de tamanho $10.000 \times r$
- Uma matriz B de tamanho $r \times 10.000$

Onde r (o *rank*) é um número muito pequeno, como 8 ou 16. A multiplicação dessas duas matrizes ($A \times B$) resulta em uma nova matriz do mesmo tamanho da original (10.000×10.000), mas que exigiu treinar apenas 160.000 parâmetros em vez de $100.000.000$.

Na prática, o LoRA reduz os parâmetros treináveis em até 99% sem perder quase nada de precisão ou capacidade! E o mais impressionante: o arquivo final do seu modelo treinado (o *adapter*) não terá 16 Gigabytes, mas sim algo em torno de 50 Megabytes, podendo ser distribuído facilmente por e-mail ou via repositórios online.

3 O Ecossistema: Hugging Face e Unsloth

Graças ao advento do LoRA, uma placa de vídeo comum (como as GPUs da série NVIDIA RTX) ou a GPU gratuita oferecida no *Google Colab* (como a T4) tornou-se suficiente para afinar um modelo fundacional poderoso. Mas como aplicamos essa arquitetura no código real?

O repositório global e padrão absoluto da indústria para modelos de Inteligência Artificial é o **Hugging Face Hub**. Ele atua como o autêntico "GitHub do Machine Learning", hospedando centenas de milhares de modelos *open-*

source (famílias Llama, Mistral, Qwen, Gemma) prontos para download imediato. É de lá que baixamos o modelo base.

Para tornar o treinamento usando LoRA ainda mais rápido e viável para o público em geral, surgiu recentemente uma biblioteca revolucionária chamada **Unsloth**.

◇ Informação: title=Por que usar o Unsloth?

O Unsloth é um projeto que reescreveu os cálculos matemáticos subjacentes das operações de treinamento em *CUDA* e *Triton* (linguagens de baixíssimo nível para placas de vídeo NVIDIA). Ele otimiza o código interno de frameworks como o Hugging Face (`transformers`) e PyTorch para extrair até a última gota de performance da memória gráfica (VRAM).
Com o Unsloth, operações que antes causavam estouro de memória (*Out of Memory* - OOM) rodam perfeitamente em 16GB ou até 8GB de VRAM, alcançando velocidades de treino de 2x a 5x mais rápidas do que o pipeline padrão e reduzindo drasticamente o consumo de energia.

4 Conclusão: Criando Seu Próprio Especialista

O Fine-Tuning transformou radicalmente a forma como encaramos o *Machine Learning*. Através de PEFT e da genialidade algébrica do LoRA, você não precisa se limitar à "inteligência fria" de um modelo generalista.

Com algumas centenas (ou milhares) de exemplos de conversas de alta qualidade estruturados em um arquivo JSON, e auxiliado pela velocidade do Unsloth, você pode pegar o modelo mais poderoso de código aberto e ensinar-lhe nuances complexas. Você pode ensiná-lo a usar gírias regionais, instruí-lo a responder apenas como um assistente jurídico baseado no Direito Civil Brasileiro, ou forçá-lo a ser um analisador de logs de servidor que só retorna respostas em formato XML validado.

O poder de construir IAs que eram restritas a megacorporaçõe agora é local, open-source e profundamente democrático.

O ciclo completo de Fine-Tuning demonstra que adaptar um LLM é como "afinar" um instrumento musical: o modelo base já sabe "tocar", mas o Fine-Tuning o ensina a tocar no tom e estilo que sua aplicação exige.

✔ Takeaways

- **Fine-Tuning** adapta os pesos de um modelo pré-treinado a um domínio específico, enquanto **RAG** injeta conhecimento externo no prompt sem alterar o modelo.
- O **LoRA** (Low-Rank Adaptation) congela os pesos originais e treina apenas matrizes adaptadoras de baixo rank ($A \times B$), reduzindo dramaticamente o custo computacional.
- O formato de dados para Fine-Tuning é JSONL com pares `system/user/assistant` — exatamente o padrão de conversa que vocês já usam na API.
- Fine-Tuning é indicado quando o modelo precisa aprender um **tom**, **formato** ou **vocabulário** específico que o Prompt Engineering sozinho não resolve.
- RAG é preferível quando o conhecimento muda frequentemente (documentos atualizados), pois evita re-treinar o modelo a cada atualização.
- O **Catastrophic Forgetting** é o risco de o modelo "esquecer" habilidades gerais ao ser fine-tunado em um domínio muito restrito.

5 Conclusão

O Fine-Tuning completa o arsenal de técnicas de personalização de LLMs: Prompt Engineering (sem alterar pesos), RAG (injeção de contexto externo) e agora Fine-Tuning (adaptação dos pesos internos). Na próxima aula, exploraremos a fronteira final da autonomia: os **Agentes Autônomos**, onde o LLM deixa de ser um oráculo passivo para se tornar um ator que planeja, decide e executa ações no mundo real.

✎ Questões: Autoavaliação

1 **[Direta]**Explique a diferença entre Fine-Tuning completo e LoRA. Por que o LoRA é viável em hardware consumer (GPU de 8GB) enquanto o Fine-Tuning completo exige clusters de GPUs?

2 **[Direta]**Em que formato os dados de treinamento devem ser preparados para Fine-Tuning via API da OpenAI? Descreva a estrutura JSONL com um exemplo.

3 **[Direta]**Compare RAG e Fine-Tuning como estratégias de personalização: quais métricas (custo, latência, frequência de atualização) favorecem cada abordagem?

4 **[Reflexiva]**Uma empresa quer que seu LLM responda exclusivamente em "juridiquês" formal, usando citações de artigos de lei. Discuta se Fine-Tuning, RAG ou a combinação de ambos seria a melhor estratégia e justifique.

5 **[Reflexiva]**O Catastrophic Forgetting pode fazer um modelo fine-tunado perder capacidades gerais (ex: matemática). Como um engenheiro pode detectar e mitigar esse problema durante o processo de treinamento?

Lab. 23: Fine-Tuning do Llama com LoRA

Capítulo em Revisão

Este conteúdo está sendo revisado. Algumas informações podem estar preliminares, conter erros de formatação ou sofrer alterações na versão final.

⋆ Objetivo do Laboratório

O objetivo desta atividade é realizar o ajuste fino (*Fine-Tuning*) de um modelo gigantesco (Llama-3-8B) para aprender um dialeto ou comportamento específico (ex: respostas no sotaque Mineirês). Utilizaremos a técnica LoRA por meio da biblioteca **Unsloth** para viabilizar o treinamento em uma GPU de nível básico no Google Colab.

6 Atividade Prática

1. Configuração do Ambiente (Google Colab)

1. Acesse o Google Colab.
2. Crie um novo Notebook e vá em **Runtime > Change runtime type**. Selecione a aceleração por hardware **T4 GPU**.
3. Instale as dependências do Unsloth (copie e cole no primeiro bloco):

```
!pip install unsloth
!pip install ‘‘unsloth[colab-new] @ git+https://github.com/unslothai/unsloth.git’’
!pip install --no-deps ‘‘xformers<0.0.27’’ ‘‘trl<0.9.0’’ peft accelerate bitsandbytes
```

2. O Código: Carregamento do Modelo e Preparo

Crie um arquivo JSON com cerca de 20 a 50 exemplos de treinamento, no formato `instrução` e `resposta`. Em seguida, carregue o modelo base e configure os adaptadores LoRA.

```
from unsloth import FastLanguageModel
import torch

max_seq_length = 2048
dtype = None # Autodetect
load_in_4bit = True # Redução massiva de uso de VRAM
```

```
# 1. Carrega o modelo base otimizado
model, tokenizer = FastLanguageModel.from_pretrained(
    model_name = ‘‘unsloth/llama-3-8b-bnb-4bit’’,
    max_seq_length = max_seq_length,
    dtype = dtype,
    load_in_4bit = load_in_4bit,
)

# 2. Configura os adaptadores LoRA
model = FastLanguageModel.get_peft_model(
    model,
    r = 16, # Tamanho do adaptador (Rank)
    target_modules = [‘‘q_proj’’, ‘‘k_proj’’, ‘‘v_proj’’, ‘‘o_proj’’,
                      ‘‘gate_proj’’, ‘‘up_proj’’, ‘‘down_proj’’,],
    lora_alpha = 16,
    lora_dropout = 0,
    bias = ‘‘none’’,
    use_gradient_checkpointing = ‘‘unsloth’’,
    random_state = 3407,
)
print(‘‘LoRA configurado com sucesso!’’)
```

3. O Treinamento (Trainer)

Carregue os seus dados e inicie o treinamento do modelo.

```
from trl import SFTTrainer
from transformers import TrainingArguments
from datasets import load_dataset

# Dataset fictício para exemplo:
# dataset = load_dataset(‘‘json’’, data_files=‘‘meus_dados_mineiros.json’’, split=‘‘tr

trainer = SFTTrainer(
    model = model,
    tokenizer = tokenizer,
    train_dataset = dataset, # Seu dataset formatado
    dataset_text_field = ‘‘text’’,
    max_seq_length = max_seq_length,
    args = TrainingArguments(
        per_device_train_batch_size = 2,
        gradient_accumulation_steps = 4,
        warmup_steps = 5,
```

```
        max_steps = 60, # Número curto para aula
        learning_rate = 2e-4,
        fp16 = not torch.cuda.is_bf16_supported(),
        bf16 = torch.cuda.is_bf16_supported(),
        logging_steps = 1,
        output_dir = ''outputs'',
    ),
)

trainer_stats = trainer.train()
```

4. Entrega via Git e Colab

1. Exporte o Notebook do Colab (`.ipynb`) que contém a prova da execução (logs de perda/loss diminuindo).
2. Adicione o arquivo no seu repositório Git: `git add fine_tuning_lora.ipynb`.
3. Crie o commit: `git commit -m ''Laboratório de Fine-Tuning LoRA Llama-3''`.
4. Faça o push para o seu branch principal.

7 Critério de Avaliação

A aprovação ocorre pela submissão do Notebook comprovando a queda na curva de Loss durante as épocas do Hugging Face Trainer e um exemplo de inferência gerada ao final do notebook demonstrando a aquisição da nova habilidade pelo modelo.

✔ Takeaways

- Você executou o **Fine-Tuning de um modelo de 8 bilhões de parâmetros** (Llama-3) usando apenas uma GPU T4 gratuita do Google Colab — graças ao LoRA e quantização 4-bit.
- O LoRA congela os pesos originais e treina apenas **matrizes adaptadoras** de baixo rank ($r = 16$), reduzindo o número de parâmetros treináveis de bilhões para milhões.
- A curva de Loss decrescente no log do Trainer é a **prova empírica** de que o modelo está aprendendo o novo comportamento.
- O resultado final (inferência com sotaque/estilo novo) demonstra que o modelo adquiriu uma habilidade **sem perder** as capacidades gerais.

◇ **Informação**

Gancho para a Próxima Aula: O Fine-Tuning ensina o modelo a “falar” de um jeito novo, mas ele continua sendo um oráculo passivo. Na próxima aula, daremos ao LLM a capacidade de **agir no mundo real**: os **Agentes Autônomos** usam Function Calling e o padrão ReAct para planejar, decidir e executar ações via APIs externas.