Reprodução do Método LoRA para Adaptação do GPT-2 na Classificação de Sentimentos usando SST-2

Pedro H. A. G. Peixinho[1], Raissa Heimann[1], Sergio L. B. Paixao[1], Victor G. de Carvalho[1], Vinícius S. O. Brito[1]

[1]Centro de Informática – Universidade Federal de Pernambuco (UFPE)
Recife – PE – Brazil

{phagp, rh, slbp, vgc3, vsob}@cin.ufpe.br

Resumo. *Este estudo investiga a eficiência da adaptação do modelo GPT-2 para análise de sentimentos no dataset SST-2, comparando três abordagens: fine-tuning completo, LoRA (Low-Rank Adaptation) e uma técnica de ajuste de camadas iniciais (PreLayer). Enquanto o fine-tuning completo atualiza todos os 124 milhões de parâmetros do modelo, o LoRA ajusta apenas 0,12% desses parâmetros. Os resultados demonstram que o LoRA não apenas reduz o consumo de memória GPU em aproximadamente 50% e o tempo de treinamento em 29%, mas também supera o fine-tuning completo em desempenho, atingindo uma acurácia de 86,93% contra 83,26%. A análise confirma que métodos de Parameter-Efficient Fine-Tuning (PEFT), como o LoRA, representam uma alternativa viável e, neste caso, superior, oferecendo um excelente balanço entre desempenho e eficiência computacional.*

Abstract. *This work investigates the efficiency of adapting the GPT-2 model for sentiment classification on the SST-2 dataset by comparing three approaches: full fine-tuning, Low-Rank Adaptation (LoRA), and a selective layer adaptation technique (PreLayer). While full fine-tuning updates all 124 million parameters, LoRA adjusts only 0.12% of them. Experimental results show that LoRA not only reduces GPU memory usage by about 50% and training time by 29%, but also outperforms full fine-tuning, achieving an accuracy of 86.93% and an F1-score of 87.42% compared to 83.26% and 83.41% for full fine-tuning. PreLayer, which trains only 0.037% of parameters, matches the performance of full fine-tuning with even lower resource consumption. These findings confirm that Parameter-Efficient Fine-Tuning (PEFT) methods such as LoRA offer an excellent trade-off between computational efficiency and performance, making them highly suitable for resource-constrained and scalable NLP applications.*

1. Introdução

Com o avanço dos modelos de linguagem de grande escala (LLMs), tornou-se cada vez mais necessário buscar métodos de adaptação que conciliem desempenho com eficiência computacional. O GPT-2, um modelo autoregressivo baseado na arquitetura Transformer [1], demonstrou-se eficaz em diversas tarefas de Processamento de Linguagem Natural (PLN). No entanto, seu *fine-tuning* completo exige altos custos computacionais e armazenamento de grandes volumes de parâmetros.

Nesse contexto, o método **LoRA (Low-Rank Adaptation)** [2] surge como uma alternativa promissora. Ao congelar os pesos originais do modelo e introduzir matrizes

de baixo posto apenas nas projeções de atenção, o LoRA permite o ajuste eficiente de LLMs, reduzindo drasticamente tanto a quantidade de parâmetros treináveis quanto o tempo necessário para o treinamento.

Este trabalho tem como objetivo reproduzir e avaliar a aplicação do LoRA no modelo GPT-2 para a tarefa de análise de sentimentos, utilizando o conjunto de dados SST-2 (Stanford Sentiment Treebank) [3]. Foram conduzidos experimentos com três configurações distintas:

- **Fine-tuning completo do GPT-2 para classificação**, com atualização de todos os parâmetros do modelo;
- **Adaptação via LoRA**, em que apenas matrizes adicionais de baixo posto são treinadas;
- **Treinamento com PreLayer** [4], uma abordagem intermediária que busca adaptar apenas camadas específicas antes dos blocos Transformer, oferecendo um equilíbrio entre custo e flexibilidade.

Além da análise de desempenho com métricas como acurácia e F1-score, este estudo também investiga aspectos de eficiência computacional, como o número de parâmetros treináveis e o tempo total de treinamento. O foco está em avaliar a viabilidade prática dessas abordagens em contextos com restrições de recursos computacionais, múltiplas tarefas simultâneas e necessidades de escalabilidade.

2. Metodologia

2.1. Base de dados

Utilizamos o **SST-2** (Stanford Sentiment Treebank) conforme disponibilizado no `GLUE benchmark` via `datasets.load_dataset("glue", "sst2")`. O conjunto fornece sentenças curtas em inglês rotuladas como `0` (negativo) ou `1` (positivo). Foi utilizada a divisão padrão de treino e validação.

2.2. Tokenização

Foi utilizado o `GPT2Tokenizer` (modelo base ``'gpt2''`) para garantir compatibilidade direta com o modelo pré-treinado GPT-2, evitando problemas de desalinhamento entre o vocabulário e os embeddings. Esse tokenizador aplica a técnica de *Byte Pair Encoding (BPE)* [5], que permite lidar bem com palavras desconhecidas e manter as sequências compactas, mesmo em textos variados.

Como o GPT-2 foi originalmente treinado para tarefas de geração de texto, ele não possui um token de preenchimento (`[PAD]`) definido. Para contornar esse problema na formação de batches com tamanhos iguais, foi necessário atribuir ao token de padding o mesmo valor do token de fim de sequência, com o comando `tokenizer.pad_token = tokenizer.eos_token`.

Também foram definidos os parâmetros `truncation=True`, `padding="max_length"` e `max_length=128`, a fim de padronizar o comprimento das sequências e evitar estouros de memória durante o treinamento. Esses ajustes garantiram que os dados fossem processados corretamente e sem erros ao longo das etapas do fine-tuning.

2.3. Modelos

Para a análise comparativa, foram utilizadas quatro abordagens distintas, todas baseadas no modelo GPT-2 (versão `gpt2 para classificação`, com 124 milhões de parâmetros), conforme disponibilizado pela biblioteca `transformers` [6]. A seguir, detalha-se a configuração de cada uma.

1. **GPT-2 sem ajuste fino (baseline inicial):** Para estabelecer uma linha de base que permita verificar se as demais abordagens promovem de fato aprendizado, foi utilizado o modelo `GPT2ForSequenceClassification` com pesos pré-treinados, mas sem qualquer etapa de ajuste fino para a tarefa de classificação de sentimentos. Esse modelo é avaliado diretamente, usando apenas a camada de classificação inicializada aleatoriamente, a fim de registrar suas métricas de desempenho antes do treinamento. A comparação entre esses valores iniciais e os obtidos após o treinamento evidencia o ganho real proporcionado por cada abordagem.
2. **Fine-Tuning Completo (FT Completo):** Esta é a abordagem tradicional de adaptação. Foi instanciado o modelo `GPT2ForSequenceClassification`, que adiciona uma camada de classificação linear no topo da arquitetura base do GPT-2. Durante o treinamento, todos os 124 milhões de parâmetros do modelo, incluindo os da camada de classificação e os do corpo principal do Transformer, foram descongelados e atualizados (`requires_grad=True`).
3. **GPT-2 com LoRA (*Low-Rank Adaptation*):** Para esta abordagem de eficiência de parâmetros (PEFT), o modelo `GPT2ForSequenceClassification` foi modificado com a técnica LoRA, utilizando a biblioteca `peft`. A configuração LoRA foi aplicada especificamente às camadas de atenção (`c_attn`) do modelo. Neste método:
    - Os pesos originais do GPT-2 foram congelados, não sendo atualizados durante o treinamento.
    - Matrizes de baixa ordem (baixo *rank*) foram injetadas nas camadas de atenção, e apenas os parâmetros dessas novas matrizes foram treinados.

    Essa técnica reduz drasticamente o número de parâmetros ajustáveis, focando a adaptação nos componentes do modelo mais relevantes para a compreensão contextual.
4. **GPT-2 com *PreLayer*:** Esta abordagem experimental foca o treinamento em um subconjunto mínimo de parâmetros localizados antes dos blocos Transformer. Especificamente, apenas as camadas de embedding de tokens (`wte`) e de embedding posicional (`wpe`) foram descongeladas e treinadas. A hipótese é que a adaptação do espaço de representação inicial das palavras e suas posições pode ser suficiente para ajustar o modelo à tarefa de classificação de sentimentos, oferecendo um método ainda mais econômico que o LoRA em termos de parâmetros treináveis, enquanto o restante do modelo (os blocos Transformer) permanece congelado.

2.4. Métricas utilizadas

Para avaliar as duas abordagens, utilizamos dois tipos de métricas: as que medem o desempenho dos modelos na tarefa e as que medem o custo para treiná-los.

2.4.1. Métricas de Desempenho

Estas métricas quantificam a qualidade dso modelos na classificação dos sentimentos das frases.

- **Acurácia:** Mede o percentual de classificações corretas realizadas pelo modelo.
- **F1-Score:** É calculado como a média harmônica entre precisão e recall e avalia a capacidade do modelo de manter um bom desempenho tanto para a classe positiva quanto para a negativa, oferecendo uma visão mais robusta que a acurácia isoladamente.

2.4.2. Métricas de Eficiência

Estas métricas mostram o custo computacional de cada abordagem.

- **Tempo de treinamento:** O tempo total, em segundos, necessário para concluir o processo de treinamento. A medição foi feita no mesmo ambiente de hardware para ambos os métodos.
- **Número de parâmetros:** A contagem dos pesos dos modelos que são atualizados durante o treinamento.
- **Gasto de Memória:** Avalia o consumo de memória de vídeo (VRAM), um indicador crítico para a viabilidade do treinamento em hardware com recursos limitados. Para uma análise detalhada, foram observadas duas componentes:
  - **Memória Alocada:** A quantidade de memória efetivamente ocupada pelos tensores do modelo, gradientes e saídas das camadas.
  - **Memória em Cache:** A memória adicional que o driver CUDA reserva para otimizar o desempenho, permitindo a rápida reutilização de blocos de memória.

  Analisar ambas as métricas oferece uma visão completa da pegada de memória de cada abordagem.

2.5. Configurações do treinamento

2.5.1. Hiperparâmetros de Treinamento

Tabela 1 resume os principais hiperparâmetros utilizados nesta primeira execução, sendo os mesmos utilizados no artigo original do LoRa.

Vale destacar que o número de épocas foi limitado a 2 devido às restrições de recursos computacionais disponíveis no ambiente de execução (Google Colab). Embora um maior número de épocas pudesse, em teoria, aprimorar o desempenho dos modelos, essa limitação foi necessária para garantir a viabilidade da execução completa de todos os métodos propostos dentro do tempo e capacidade de processamento disponíveis. Optou-se, portanto, por manter o mesmo número de épocas para todas as abordagens a fim de preservar a equidade na comparação dos resultados.

2.5.2. Treinamento e Registro

O treinamento foi conduzido em GPU (ambiente notebook). O rastreamento de métricas foi feito localmente; o *logging* para Weights & Biases foi desativado explicitamente com

Tabela 1. Hiperparâmetros de treinamento.

	FT Completo e PreLayer	LoRA
Épocas	2	2
Batch size (treino/val)	16 / 16	16 / 16
Learning rate	2e-4	2e-4
Weight decay	0.01	0.01
Dropout LoRA	—	0.05
Rank r	—	4
α (LoRA scaling)	—	16
Camadas-alvo LoRA	—	`c_attn`*

** A camada `c_attn` do GPT-2 é uma projeção linear única que produz Q, K e V. Adaptá-la com LoRA é equivalente a modificar as projeções de Q e V em modelos onde essas camadas são separadas.*

`report_to="none"` para evitar prompts de autenticação durante a execução automatizada.

2.6. Diferenças em Relação ao Estudo Original

Embora este trabalho seja inspirado na metodologia do artigo do LoRA [2], ele se diferencia em escopo e configuração, com o objetivo de avaliar a técnica em um cenário específico e com recursos computacionais limitados. As principais distinções são:

- **Modelo Avaliado:** Neste estudo, utilizamos o `GPT2ForSequenceClassification`, ou seja, a variante do GPT-2 já adaptada para tarefas de classificação, com uma camada linear no topo para previsão de rótulos. No trabalho original, a aplicação do LoRA sobre o GPT-2 foi feita no modelo base convencional (`GPT2LMHeadModel`), voltado para geração de texto.
- **Dataset e Tarefa:** Este estudo foca em uma única tarefa de classificação de sentimentos, utilizando o dataset **SST-2**. O artigo de referência aplicou o LoRA em uma gama mais ampla de tarefas, como as do benchmark GLUE, e em tarefas de geração de texto (E2E NLG).
- **Duração do Treinamento:** Devido às restrições do ambiente experimental, os modelos foram treinados por apenas 2 épocas, um regime de treinamento mais curto em comparação com os utilizados no estudo original.

3. Resultados

A seguir, apresentam-se os resultados quantitativos obtidos, com foco nos *trade-offs* entre desempenho (Acurácia e F1-Score) e custo computacional (parâmetros treináveis, uso de memória e tempo de treinamento) para cada abordagem avaliada.

3.1. Desempenho dos modelos

As métricas de acurácia e F1-Score estão resumidas na Tabela 2.

O modelo *baseline* sem ajuste fino, avaliado diretamente antes de qualquer etapa de treinamento, apresentou desempenho significativamente inferior às demais estratégias, com acurácia de apenas 50,46% e F1-Score de 21,45%. Esse resultado evidencia que,

Tabela 2. Comparação de desempenho (Acurácia e F1-Score).

Método	Acurácia	F1-Score
Sem Fine-Tuning	0.5046	0.2145
Fine-Tuning Completo	0.8326	0.8341
LoRA	**0.8693**	**0.8742**
PreLayer	0.8314	0.8372

para a tarefa considerada, a configuração inicial do modelo não dispõe de parâmetros especializados para a classificação de sentimentos, servindo apenas como referência para quantificar o ganho obtido após o treinamento.

O *Fine-Tuning* completo foi utilizado como *baseline* de desempenho, obtendo Acurácia de 0,8326 e F1-Score de 0,8341. O método PreLayer apresentou desempenho estatisticamente equivalente ao *baseline* (Acurácia: 0,8314; F1-Score: 0,8372). O LoRA superou todas as demais abordagens, alcançando as melhores métricas (Acurácia: 0,8693; F1-Score: 0,8742) — mesmo com redução drástica no número de parâmetros treináveis.

De forma geral, os resultados mostram que abordagens *Parameter-Efficient Fine-Tuning* (PEFT) não apenas reduzem custo computacional, mas também podem superar estratégias de ajuste fino completo.

3.2. Eficiência computacional

A Tabela 3 apresenta a comparação do número de parâmetros treináveis, sua fração em relação ao total do modelo e o tempo total de treinamento.

Tabela 3. Comparação de custo computacional (parâmetros e tempo).

Método	Parâmetros Treináveis	% do Total	Tempo de Treino (s)
Fine-Tuning Completo	124,441,344	100%	3492.40
LoRA	148,992	0.1197%	2464.95
PreLayer	46,080	0.0370%	2662.89

No *Fine-Tuning* completo, todos os parâmetros do GPT-2 foram atualizados, totalizando cerca de 124 milhões de parâmetros (100% do total). Essa configuração demandou o maior tempo de treinamento (3492,40 s) e o maior consumo de memória.

O LoRA atualizou apenas 148.992 parâmetros (0,1197% do total), enquanto o PreLayer treinou 46.080 parâmetros (0,0370%). Apesar dessa redução drástica no número de parâmetros, o tempo de treinamento não caiu proporcionalmente, pois o custo do *forward pass* permanece dominado pelo backbone congelado. Ainda assim, houve ganho de desempenho e expressiva economia de recursos.

A Tabela 4 detalha o consumo de memória GPU durante o treinamento.

Observa-se que LoRA e PreLayer consomem aproximadamente 74% menos memória alocada que o ajuste fino completo, além de redução significativa na memória em cache. Essa característica torna as abordagens *Parameter-Efficient Fine-Tuning* (PEFT) especialmente atrativas em ambientes com restrição de recursos.

Tabela 4. Consumo de memória GPU durante o treinamento.

Configuração	Memória Alocada	Memória em Cache
FT completo	1.89 GB	4.12 GB
LoRA	0.49 GB	2.50 GB
PreLayer	0.49 GB	2.34 GB

A menor quantidade de parâmetros treináveis torna o LoRA extremamente vantajoso em contextos onde escalabilidade, agilidade e economia de recursos são essenciais. Destacamos algumas aplicações práticas onde essa eficiência é particularmente útil:

- **Cenários multi-task:** Em pipelines que envolvem múltiplas tarefas (e.g., classificação, sumarização, QA), é possível treinar cabeçalhos LoRA específicos para cada tarefa, mantendo o backbone fixo. Isso permite alternar rapidamente entre tarefas sem reprocessar todo o modelo, como exemplificado em sistemas de assistência virtual que precisam alternar entre domínios como jurídico, médico e educacional.
- **MLOps com múltiplos clientes:** Em contextos de *SaaS* ou serviços personalizados (e.g., chatbots corporativos), é possível manter um único modelo-base congelado e treinar instâncias LoRA leves para cada cliente, facilitando isolamento de versões, atualização individualizada e economia em armazenamento e inferência. Por exemplo, uma plataforma de atendimento pode oferecer modelos customizados para empresas distintas sem necessidade de replicar todo o backbone.
- **Experimentação rápida em hardware limitado:** Pesquisadores e desenvolvedores podem realizar testes com variações de tarefas ou domínios sem acesso a GPUs de alto desempenho. A leveza dos cabeçalhos LoRA permite rodar experimentos mesmo em notebooks com GPU modesta ou ambientes de nuvem com recursos restritos (e.g., Colab, EC2 de entrada).
- **Versionamento e reprodutibilidade:** Como apenas os pesos adicionais (as "diffs") do LoRA precisam ser armazenados, o controle de versões se torna mais simples e leve. Isso facilita reverter a versões anteriores, comparar variantes de modelos e auditar ajustes realizados — algo especialmente valioso em ambientes regulados, como saúde e finanças.

4. Discussões

Os resultados deste estudo apresentam um ponto de análise central e contraintuitivo: a abordagem LoRA, treinando apenas 0,12% dos parâmetros, não apenas se igualou, mas superou o desempenho do fine-tuning completo. Esta seção se aprofunda nas possíveis causas para esse fenômeno, no impacto das decisões metodológicas e na interpretação dos resultados das abordagens experimentais.

4.1. A Estabilidade do Treinamento e a Preservação do Conhecimento

A superioridade do LoRA (acurácia de 86,93%) sobre o Fine-Tuning Completo (83,26%) pode ser atribuída à estabilidade de seu processo de adaptação. O fine-tuning completo, ao atualizar todos os 124 milhões de parâmetros, pode sofrer de uma dinâmica de treinamento mais instável.

Em contrapartida, o LoRA adota uma abordagem mais conservadora e menos impactante. Ao congelar os pesos originais e introduzir apenas matrizes aditivas de baixo posto, ele garante que o núcleo de conhecimento do modelo permaneça intacto. A adaptação ocorre de forma mais controlada e estável, construindo sobre a base existente em vez de modificá-la radicalmente. Essa abordagem permite uma convergência mais eficiente para um ponto de ótimo superior na tarefa específica de análise de sentimentos.

4.2. Impacto da Configuração de Treinamento e Hiperparâmetros

A configuração do treinamento é um fator crítico para a interpretação dos resultados. O uso de apenas duas épocas, uma decisão pragmática devido a restrições de recursos, favorece métodos que convergem rapidamente. O LoRA demonstrou ser um desses métodos, alcançando um pico de desempenho superior em um curto período. É plausível que, com mais épocas e um ajuste fino da taxa de aprendizagem (tipicamente menor para fine-tuning completo, como 2×10^{-5}, para evitar instabilidade), o modelo de Fine-Tuning Completo pudesse eventualmente alcançar ou até superar o LoRA. No entanto, o fato de o LoRA entregar um resultado superior com um orçamento computacional menor (29% menos tempo) é, por si só, uma conclusão poderosa sobre sua viabilidade prática.

5. Conclusão

Este estudo demonstrou, de forma quantitativa e qualitativa, que o método LoRA representa uma alternativa altamente eficiente ao fine-tuning completo para a adaptação de grandes modelos de linguagem, como o GPT-2, na tarefa de classificação de sentimentos. Ao reduzir em mais de 99,8% o número de parâmetros treináveis e consumir aproximadamente metade da memória GPU, o LoRA alcançou desempenho superior em acurácia e F1-score quando comparado ao ajuste fino tradicional.

Os resultados também evidenciaram que abordagens intermediárias, como o PreLayer, podem oferecer um equilíbrio entre custo computacional e desempenho, aproximando-se da performance do fine-tuning completo com um custo mínimo em termos de recursos. Esse achado reforça a importância de explorar estratégias de textitParameter-Efficient Fine-Tuning (PEFT) em cenários com restrições de hardware ou necessidade de escalabilidade.

Além disso, verificou-se que a leveza do LoRA possibilita aplicações práticas em contextos como sistemas multi-tarefa, serviços personalizados para múltiplos clientes e experimentação rápida em ambientes com recursos limitados. Essas características tornam o método particularmente atraente para pipelines de MLOps e desenvolvimento de soluções comerciais escaláveis.

Como trabalhos futuros, propõe-se expandir a investigação para modelos de maior porte, como GPT-J [7] ou LLaMA [8], e avaliar o impacto do LoRA em tarefas mais complexas e de diferentes domínios. Também seria relevante explorar a combinação do LoRA com outras técnicas de PEFT e mecanismos de regularização, buscando otimizar ainda mais o balanço entre eficiência e desempenho.

Em síntese, os experimentos realizados confirmam que o LoRA não apenas preserva a qualidade do modelo base, mas também amplia sua viabilidade prática, oferecendo um excelente custo-benefício para adaptações especializadas de LLMs.

Referências

[1] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is All You Need. In *Advances in Neural Information Processing Systems*, volume 30, 2017.

[2] Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-Rank Adaptation of Large Language Models. *arXiv preprint arXiv:2106.09685*, 2021.

[3] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. Recursive Deep Models for Semantic Compositionality Over a Sentiment Treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics, 2013.

[4] Xiang Lisa Li and Percy Liang. Prefix-Tuning: Optimizing Continuous Prompts for Generation. *arXiv preprint arXiv:2101.00190*, 2021.

[5] Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural Machine Translation of Rare Words with Subword Units. *arXiv preprint arXiv:1508.07909*, 2016.

[6] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics, 2020.

[7] Ben Wang and EleutherAI. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. *GitHub*, 2021. Disponível em: https://github.com/kingoflolz/mesh-transformer-jax.

[8] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and Efficient Foundation Language Models. *arXiv preprint arXiv:2302.13971*, 2023.