

Exploring LoRA Variants With GPT-2

Stanford CS224N Default Project

George Song

Department of Computer Science
Stanford University
gsong17@stanford.edu

Justin Choo

Department of Computer Science
Stanford University
oliver88@stanford.edu

Abstract

Fine-tuning large language models for downstream tasks requires retraining all parameters, creating a significant bottleneck for compute resources. Low-Rank Adaptation (LoRA) addresses this by injecting trainable low-rank matrices into Transformer layers while freezing the pre-trained weights. In this project, we re-implement GPT-2 and explore how LoRA can mimic the performance of full fine-tuning on a paraphrase detection task. As a modification of standard LoRA, we assess the importance of each layer using ablation tests, and then use this information to dynamically allocate rank across our 12 Transformer layers. We examine the efficacy of this dynamically-allocated version of LoRA, in addition to other variations on standard LoRA. We find that LoRA, dynamic LoRA, and DoRA slightly underperform relative to full fine-tuning, while Init[AB] achieves the same performance of full fine-tuning. These results suggest that LoRA and its variants are viable alternatives to full fine-tuning, and that the initialization of weights in the LoRA regime may be critical to model performance.

1 Key Information to include

- Mentor: Alisa Levin
- External Collaborators (if you have any): No
- Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
George Song	1
Justin Choo	3
Total pooled	4
Team size	2
Late days applied for the report	2

2 Introduction

While modern frontier LLMs generally adhere to the philosophy that more parameters are necessary to achieve more powerful models, such levels of compute are not only inaccessible to the vast majority of researchers, but also unsustainable from an environmental perspective. Even for a relatively modest model like GPT-2, this means retraining up to 774 million parameters for each task [1], and the problem only grows worse for models like GPT-3 with 175 billion parameters [2]. Parameter-efficient fine-tuning (PEFT) methods aim to reduce this cost, but prior techniques such as adapter layers and prefix tuning introduce inference latency, increase optimization difficulty, or decrease model performance compared to full fine-tuning baselines. Low-Rank Adaptation (LoRA) [3] has emerged as a prominent PEFT approach, freezing pretrained weights and training only small low-rank matrices

injected into Transformer layers. However, standard LoRA applies the same rank uniformly across all layers, despite prior work suggesting that different weight matrices have varying importance for downstream tasks [4]. A growing number of LoRA variants have also been proposed. Init[AB] [5] and Weight-Decomposed Low-Rank Adaptation (referred to as DoRA) [6] are two variants that claim to improve upon standard LoRA through modified initialization or architecture, but it remains unclear how much these gains hold across tasks and settings.

The goal of our project is to explore whether LoRA variants can outperform standard LoRA on paraphrase detection, and whether smarter allocation of parameters across GPT-2’s 12 Transformer layers can match full fine-tuning performance. We implement standard LoRA, Init[AB], DoRA, and two dynamic per-layer rank allocation strategies, comparing them all against each other and against full fine-tuning on paraphrase detection. To guide our dynamic allocation strategies, we conduct layer-wise ablation studies that measure each layer’s contribution to task performance. Separately, we fine-tune GPT-2 on sentiment classification and sonnet generation to evaluate the base model across diverse tasks.

Our key finding is that layer importance for paraphrase detection is surprisingly uniform across all 12 Transformer layers, meaning that dynamic rank allocation does not help. In addition, most of the methods we tested (DoRA, dynamic LoRA, and standard LoRA) slightly underperformed relative to vanilla GPT-2 on paraphrase detection, with the exception of Init[AB], which was the only method to match full fine-tuning performance.

3 Related Work

Hu et al. [3] introduced LoRA, motivated by the observation that pretrained models have low intrinsic dimensionality [7] and that task-specific weight updates can be captured by low-rank structures. They showed that on GPT-2, LoRA matched or outperformed full fine-tuning on natural language generation benchmarks with only 0.35 million trainable parameters. However, LoRA applies the same rank uniformly across all layers, leaving open the question of whether a non-uniform allocation could do better.

Zhang et al. [4] addressed this with AdaLoRA, which dynamically adjusts the rank of each layer during training based on learned importance scores. Our dynamic LoRA approach is related but uses a simpler fixed allocation based on empirical ablation results rather than learned importance scores. Hayou et al. [5] proposed Init[AB], which initializes both low-rank matrices randomly instead of zero-initializing B . Liu et al. [6] proposed DoRA, which separates magnitude and direction components of the weight update.

Most relevant to our project, Lee et al. [8] recently conducted a large-scale re-evaluation of LoRA variants including DoRA and Init[AB] on mathematical reasoning and code generation tasks. They found that different methods favor different learning rate ranges, but once learning rates are properly tuned, all methods reach similar peak performance (within 1-2%). Their study focused on models with up to 7B parameters on math and coding tasks. Our work examines a similar question on a smaller model (GPT-2) and on language tasks.

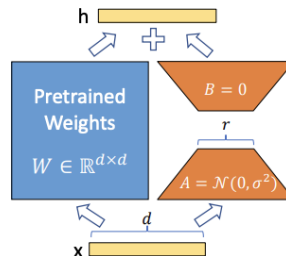


Figure 1: LoRA reparametrization [3].

4 Approach

4.1 Base model.

We used GPT-2 as a base model for our experiments. GPT-2 uses byte-pair encoding which maps to a trainable embedding layer. The model uses 12 Decoder Transformer layers, each of which uses masked multiheaded attention with 768-dimensional hidden states and 12 attention heads. We reimplemented the core components of GPT-2 from scratch, including masked multi-head self-attention, the GPT-2 Transformer layers, the token and positional embedding function, and the Adam optimizer [9]. Finally, we loaded pre-trained weights from HuggingFace.

4.2 Sentiment Classification.

We added a classifier layer on top of the base model with dropout in order to generate predictions for the most likely sentiment of the provided example. The input to the classifier layer is the hidden state of the last non-padding token in the input sequence. The number of output logits is consistent with the number of possible labels in the dataset.

4.3 Paraphrase Detection.

Each input is formatted as a query in the form, "*Is 's1' a paraphrase of 's2'? Answer 'yes' or 'no':*" We then projected the hidden state of the last non-padding token to token space, where we extracted the logits of the tokens corresponding to 'yes' and 'no'. Due to the size of the training set, as well as limitations on compute, we took a random subset of 50,000 examples from the training set to train the model on each epoch.

4.4 Sonnet Generation.

We train GPT-2 to perform autoregressive language modeling on Shakespearian sonnets and then generate the remaining 11 lines of a sonnet, conditioned on the first 3 lines.

Training. We added `ignore_index` to the cross-entropy loss to mask out padding tokens, since the original loss computed over all tokens including padding. We implemented early stopping that halts training if the loss does not improve by more than 0.01 for 3 consecutive epochs, saving the best model checkpoint based on training loss. We experimented with unfreezing the last 4, 8, and all 12 Transformer layers, finding that unfreezing all 12 performed best. Early dev runs were affected by an evaluation bug where we scored against the test file (`sonnets_held_out.txt`) instead of the dev file (`sonnets_held_out_dev.txt`); correcting this alone improved chrF from 20.8 to 28.8 with no other changes.

Generation. Our generation approach uses top-k filtering with $k = 50$ on top of the existing top-p sampling to reduce low-probability noise. We added a line counter that stops generation after 11 lines and increased `max_length` from 128 to 200 to allow enough tokens for a full sonnet.

We observed that the model frequently produced severe repetition loops (e.g. repeating a single word dozens of times), so we implemented a repetition penalty that multiplies negative logits (or divides positive logits) by 2.0 for any token appearing in the last 20 generated tokens. We chose a local window rather than a global penalty because penalizing all previously seen tokens decreased quality. We tested penalty values of 1.2, 1.5, and 2.0, and found that only 2.0 reliably broke the loops.

We also noticed that the model generated text as a single line of prose with no line breaks, despite learning to predict newlines during training. To address this, we added a newline boost that increases the logit for the newline token based on how many tokens have been generated since the last newline. The boost begins after 6 tokens and increases linearly up to a maximum of 8.0, applied before top-k filtering. Applying the boost after top-k had no effect since the newline token was rarely in the top-k set, confirming that placement before filtering is essential.

4.5 Extensions.

We added several modifications to the model to test the efficacy of different training regimes in lieu of full fine-tuning.

LoRA. First, we implemented LoRA by defining two low-rank matrices, $A \in \mathbb{R}^{r \times d}$, $B \in \mathbb{R}^{d \times r}$ with $r = 4$, $d = 768$ for each of the attention matrices Q , K , and V at each attention layer. This yields 221,184 trainable LoRA parameters across all 12 layers (approximately 0.18% of GPT-2 Small's 124M total parameters). A is initialized with a Kaiming uniform distribution, while B is initialized as a 0-matrix. On each forward pass, the input is passed through A and B and then added to the output of the frozen weights. During backpropagation, A and B are updated while the pre-trained weights stay frozen. As defined by Hu et al. [3], the output h is defined as:

$$h = W_0x + \Delta Wx = W_0x + BAx \quad (1)$$

Dynamic LoRA. We then ran ablation tests [10] using the paraphrase detection task to determine which layers contributed most to model performance under the LoRA regime. Specifically, we retrained the model while excluding A and B matrices from specific layers and then evaluated the impact of the ablation on the model’s dev accuracy. Due to compute limitations, we discretized the model layers into 3-layer blocks and ran 4 total ablation tests.

Using the results of our ablation tests, we then implemented dynamic LoRA by allocating rank capacity to each of the 12 layers based on their impact on model performance. That is, we constrained the total rank budget across all layers to $\sum_{i=1}^{12} r_i = 48$, matching the total rank of the standard LoRA baseline ($r = 4$ per layer), so that the number of trainable LoRA parameters remains roughly constant.

We tested two allocation strategies. The first, which we call ablation, assigns $r_i = 5$ to the layers that showed the greatest impact when ablated and $r_i = 3$ to the remaining layers. The second strategy, later layers, is based on the heuristic that later layers in a transformer encode more task-specific information [11]. It assigns $r_i = 2$ to layers 0–3, $r_i = 4$ to layers 4–7, and $r_i = 6$ to layers 8–11. Both strategies let us test whether non-uniform rank allocation improves performance over standard LoRA.

DoRA. We also implemented DoRA, which decomposes each pretrained weight matrix into a magnitude component and a direction component. This requires the definition of a learnable magnitude vector $m \in \mathbb{R}^d$, initialized from the pretrained weight row norms, in addition to the A, B matrices. On each forward pass, the A, B matrices are multiplied and added to the pre-trained weights. This sum is normalized along the output dimension and then scaled by m , and this matrix defines the transformation for the input. As defined by Liu et al. [6], the output h is defined as:

$$h = m \cdot \frac{W_0 + BA}{\|W_0 + BA\|} x \quad (2)$$

where the denominator normalizes along the output dimension. We detach the norm from the computation graph during backpropagation to reduce memory usage.

Init[AB]. Finally, we implemented Init[AB], which is identical to standard LoRA except that B is also initialized with the Kaiming uniform [5].

4.6 Baselines.

To find reasonable hyperparameters for our experiments, we fully fine-tuned the base model on a sentiment detection task. To establish baseline performance levels, we fully fine-tuned the base model on our two downstream extension tasks (paraphrase detection and sonnet generation). We then trained the base model using LoRA and its variants, as described above, on the paraphrase detection task to evaluate their efficacy. We chose to compare the training regimes using the paraphrase detection task given the availability of training data.

4.7 Authorship.

Aside from the provided starter code, we implemented all code, including LoRA, both dynamic LoRAs, DoRA, and Init[AB], from scratch.

5 Experiments

5.1 Data

We use four datasets. The SST dataset contains 11,855 movie review sentences with 5-class sentiment labels (8,544/1,101/2,210 train/dev/test) [12]. The CFIMDB dataset contains 2,434 movie reviews with binary sentiment labels (1,701/245/488). The Quora dataset contains over 400,000 question pairs labeled as paraphrases or not (141,506/20,215/40,431 total). We take a random 50,000 example subset of the training data on each epoch. The Sonnet dataset contains 154 Shakespearean sonnets (143/12 train/test).

5.2 Evaluation method

For sentiment classification and paraphrase detection, we use accuracy, defined as the number of correctly classified examples out of the total examples shown. For sonnet generation, we use the CHRF score, which evaluates how closely the generated text follows Shakespeare’s actual language distribution at the n -gram level.

5.3 Experimental details

To find optimal hyperparameters, we searched over fine-tuning mode (last-linear-layer vs. full-model), learning rate $\eta \in \{1e-5, 2e-5, 1e-4, 1e-3\}$, batch size $B \in \{8, 16, 32, 64\}$, dropout rate $p \in \{0.1, 0.2, 0.3\}$, and number of epochs $\in \{5, 8, 10\}$ for the sentiment classification task. Our best configuration used full-model fine-tuning on GPT-2 Small with batch size 32, learning rate 1e-5, dropout 0.3, and 5 epochs. Training was run on an NVIDIA A100 GPU via Google Colab.

For paraphrase detection, we used full-model fine-tuning on GPT-2 Medium with batch size 32, learning rate 1e-5, and 8 epochs. We used GPT-2 Medium for our best vanilla baseline submission. All LoRA variant experiments were run on GPT-2 Small to keep comparisons consistent, with a separate GPT-2 Small full fine-tuning baseline (0.861 dev accuracy) reported in Table 2. All training for paraphrase detection was run on an NVIDIA A100 GPU via Google Colab. Total training time was about 35 mins.

For sonnet generation, we tested temperatures of 0.3, 0.5, and 0.7. A temperature of 0.3 amplified degenerate patterns when all layers were unfrozen (chrF dropped from 19.2 to 17.8 compared to 8 layers unfrozen at the same temperature). Our best configuration used full-model fine-tuning on GPT-2 Small with batch size 32, learning rate 1e-5, temperature 0.7, and 20 epochs on a T4 GPU via Google Colab. Total training time for 20 epochs was about 20 mins.

For all LoRA variants (standard LoRA, both dynamic LoRAs, DoRA, and Init[AB]), we ran on on GPT-2 Small with batch size 32, learning rate 1e-5, and 8 epochs on an NVIDIA A100 GPU via Google Colab.

5.4 Results

To obtain our scores for the leaderboard, we trained vanilla GPT-2 on the three downstream tasks. We used the small version of the model for sentiment classification and sonnet generation, and we used the medium version for paraphrase detection. We were able to meet or exceed the baseline scores on all tasks.

Task	Fine-tune Mode	Dev Acc	Test Acc	Dev CHRF	Test CHRF
Sentiment (SST)	last-linear-layer	0.457	-	-	-
Sentiment (SST)	full-model	0.511	-	-	-
Sentiment (CFIMDB)	last-linear-layer	0.853	-	-	-
Sentiment (CFIMDB)	full-model	0.98	-	-	-
Paraphrase	full-model	0.888	0.888	-	-
Sonnet	full-model	-	-	42.738	42.437

Table 1: *Performance of vanilla GPT-2 baselines across tasks.* Leaderboard scores are in **bold**.

We ran 4 ablation tests, grouping the 12 layers of the model into four 3-layer blocks (Figure 3). We used the paraphrase detection task for evaluation. Each of the ablation tests yielded roughly the same model performance. Removing layers 0/1/2 and layers 6/7/8 yielded a dev accuracy of 0.857, and removing layers 3/4/5 and layers 9/10/11 yielded a dev accuracy of 0.856. It is possible that the results reflect small variations in layer importance. However, the fluctuation in dev accuracies may have resulted from the rounding of slightly different values, as accuracy was only calculated the three decimal places. Interestingly, each of the ablation tests resulted in a higher dev accuracy than standard LoRA. This may indicate possible overfitting when rank is allocated to all layers.

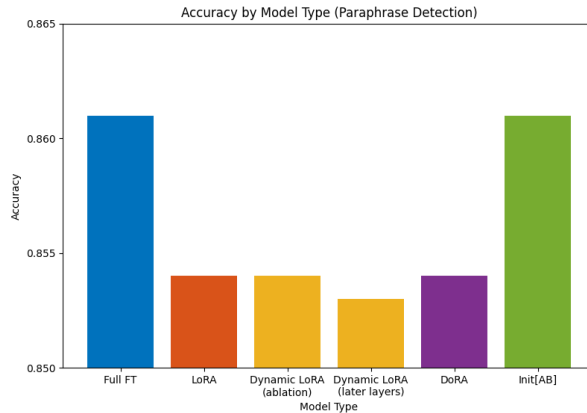


Figure 2: *Full fine-tuning vs. LoRA*. LoRA, dynamic LoRA, and DoRA approximate the performance of full fine-tuning, yet slightly underperform. Init[AB] achieves the same performance as full fine-tuning.

We ran the paraphrase detection task for all our models (Table 2, Figure 2) with consistent hyperparameters for all runs. Full-finetuning with GPT-2 Small achieved a dev accuracy of 0.861, which represented our target for our other training regimes. LoRA, dynamic LoRA (ablation), and DoRA achieved dev accuracies of 0.854 and dynamic LoRA (later layers) achieved a dev accuracy of 0.853, while Init[AB] achieved the target dev accuracy of 0.861. The general performance levels of LoRA and its variants aligned with our expectations. We anticipated that LoRA would be able to achieve similar performance levels as full-finetuning, and the lowest LoRA accuracy was within 0.01 of the target accuracy. However, the relative performance of LoRA to its variants was unexpected. We predicted that all four variants would outperform the vanilla version of LoRA, but only Init[AB] achieved a higher dev accuracy.

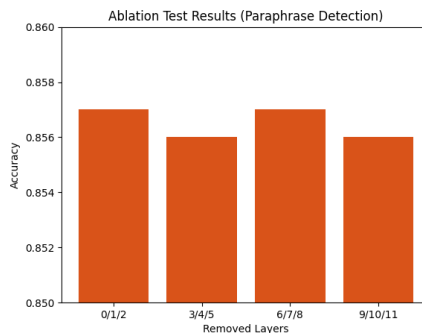


Figure 3: *Ablation tests*. Labels on the x-axis indicate the indices of the layers from which A , B matrices were removed. Layer importance is roughly equivalent across all layers.

Model	Dev Acc
GPT-2 Vanilla (full-model)	0.861
LoRA	0.854
Dynamic LoRA (ablation)	0.854
Dynamic LoRA (later layers)	0.853
DoRA	0.854
Init[AB]	0.861

Table 2: *Paraphrase detection dev accuracy across LoRA variants*. Non-dynamic LoRA variants use rank $r = 4$. All models use batch size 32, learning rate $1e-5$, and 8 epochs.

6 Analysis

6.1 Sentiment Classification

We found that the last-linear-layer approach performed noticeably worse than full-model fine-tuning on both SST and CFIMDB, which aligned with our expectations. This result shows that early layers

contribute meaningfully to the task performance. For SST, full fine-tuning improved dev accuracy by 0.054, and for CFIMDB, dev accuracy improved by 0.143, suggesting that fine-tuning all layers yields a modest but significant improvement.

6.2 Paraphrase Detection and LoRA Variants

We reviewed several examples from the dev predictions and found that the model was able to handle tricky cases well. For instance, it correctly distinguished between “How do I play 4K videos on a Samsung smart TV?” and “How can I convert and play .UM4 (Samsung 4K TV content) on my Sony 4K TV?” which share topic-level similarity and common tokens but ask different questions. It also correctly rejected pairs with very similar sentence structure but different meanings, such as “What do you do when you fall in love with your best friend?” versus “What would you do if you’re falling in love with your cousin?” However, the model did misclassify some instances, especially when the difference in phrases was nuanced. For example, the model classified the two phrases “Why computer vision is hard?” and “Why computer vision is computationally hard?” as valid paraphrases of each other, when they were not labeled as such in the dataset. The second question is related to but more specific than the first; our model may not be able to identify this type of nuance. The model also incorrectly labeled the two phrases “What was life like before and after the Industrial Revolution?” and “What was life like before the Industrial Revolution?” as valid paraphrases. Here, the second phrase is a component of the first one, which the model fails to recognize.

Our layer ablation studies showed that skipping LoRA on any group of three layers produced nearly identical results, with all four buckets falling within 0.001 of each other. This was surprising because we expected certain layers to be more important than others for paraphrase detection. The uniformity explains why both of our dynamic rank allocation strategies failed to beat standard LoRA, because if every layer contributes roughly equally, there is no benefit to redistributing rank budget.

None of the LoRA variants outperformed vanilla GPT-2 on paraphrase detection, with the exception of Init[AB], which was the only method to match full fine-tuning. We believe Init[AB] succeeds because starting with $BA \neq 0$ allows the model to begin adapting from the first update, whereas the standard zero initialization means the LoRA contribution is exactly zero at the start and must build up before it does anything useful. This is worth noting because Init[AB] is the simplest change out of all the variants we tested. It requires no architectural modification (unlike DoRA, which adds a magnitude vector and a normalization step) and no task-specific tuning (unlike dynamic LoRA, which requires ablation results to inform the rank distribution). It is just a one-line change to the initialization of B .

DoRA performed about the same as standard LoRA despite its more complex architecture. We do not have a clear explanation for why it did not help on this task, though Lee et al. [8] similarly found that DoRA’s gains over standard LoRA were much smaller than originally reported once learning rates were properly tuned.

These findings are also consistent with John Schulman’s guest lecture for CS224N, where he noted that his team has explored several LoRA variants and found the differences were minimal once learning rates were properly swept [13].

6.3 Sonnet Generation

For sonnet generation, we discovered that training quality and generation quality are not necessarily correlated. The training loss decreased steadily across all 20 epochs, yet the model still failed to produce newline tokens during generation. This means the model learned to predict newlines, but did not assign them high enough probability during autoregressive sampling for them to actually be selected. Our generation interventions addressed this directly, producing our largest single improvement without any retraining. This suggests that for small-dataset generation tasks, the generation strategy can matter far more than the training setup.

We show the effects of our generation improvements by comparing output from Sonnet 5. Before our interventions, given the prompt “Thou blind fool, Love, what dost thou to mine eyes / That they behold and see not what they see? / They know what beauty is, see where it lies,” the model generated the following as a single unbroken block of text:

with the same day’s tied by virtue of thyself that your fair woe. I am in my

love; yea their sweetest part for me as a woman or other than others ruined
his thoughts do nothing else there are no longer so far more thine eye-footed
on thee should be worthiness hath been able!

This is a single run-on sentence with incoherent token like “stied” and “eye-footed.” After our changes, the same prompt for Sonnet 5 produced:

I hold this false excuse: "Why didst thou
Make me so poor? Thou art rich in abundance,
and I shall be;
And thus dost make me lose all hope of love.
And for this cause thou art my friend
And I believe thee to be true, but not untrue.

While neither version follows the expected rhyme scheme and there is still some run-on behavior, the output now has clearer line structure and reads more as coherent Shakespearean English.

7 Conclusion

Our project demonstrates that LoRA and its variants are viable alternatives to full fine-tuning for at least some language-related tasks, and that the initialization of weights in the LoRA regime may be critical to model performance. On our paraphrase detection task, all LoRA variants performed within 0.01 of full fine-tuning accuracy, and Init[AB] matched the performance of full fine-tuning. Our ablation tests yielded nearly uniform results, and dynamically allocating rank across layers did not improve model performance, suggesting that layer-wise modifications may not be particularly useful. However, given the improvement in performance with Init[AB], the choice of initialization for the A , B matrices may lead to meaningful gains in model performance.

We learned that model behavior can be unpredictable, and that expectations may be subverted in practice. We hypothesized that running ablation tests and constructing a dynamic LoRA would improve model performance, but it turned out that each layer was roughly equal in importance.

We achieved implementations of five different fine-tuning regimes – LoRA, two dynamic LoRAs, DoRA, and Init[AB] – from scratch. This gave us full control over the mechanics of their implementations. We evaluated the relative performance of these regimes and contributed to the understanding of the impact of design choices on LoRA performance.

Our project was limited primarily by compute resources. Google Cloud VMs were almost always unavailable, so we purchased subscriptions to Colab Pro out-of-pocket. We were unable to run all 12 ablation tests as originally planned and instead grouped them into four 3-layer blocks. Moreover, we were unable to explore whether other hyperparameters might further improve the performance of LoRA, which prior work has suggested may be useful [14]. Our results are also limited by the fact that the paraphrase detection task evaluates binary outputs, so it is unclear how LoRA would perform on more open-ended tasks like sonnet generation.

Future directions of research may run more granular ablation tests, i.e. omitting LoRA matrices from each individual layer as opposed to grouping them together, in order to gain a more detailed understanding of how each layer contributes to model performance. We may also conduct hyperparameter sweeps across learning rate and batch size, which has been shown to be useful for optimizing LoRA on other tasks [8, 14]. If more compute is available, future studies may also explore LoRA with higher rank, as well as the impact of LoRA on larger versions of GPT-2.

Team Contributions

While we both contributed in some way to each task, Justin coded most of the GPT-2 base implementation, sentiment classification, paraphrase detection. George improved upon the paraphrase detection, and then coded sonnet generation. For the extension, Justin implemented standard LoRA, conducted 3/4 of the ablation tests, and added Init[AB]. George conducted the last ablation test and implemented both dynamic LoRAs and DoRA.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. *OpenAI*, 2019. Accessed: 2024-11-15.
- [2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.
- [3] 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, 2021.
- [4] Qingru Zhang, Minshuo Chen, Alexander Bukharin, Nikos Karampatziakis, Pengcheng He, Yu Cheng, Weizhu Chen, and Tuo Zhao. Adalora: Adaptive budget allocation for parameter-efficient fine-tuning, 2023.
- [5] Soufiane Hayou, Nikhil Ghosh, and Bin Yu. The impact of initialization on lora finetuning dynamics. *arXiv preprint arXiv:2406.08447*, 2024.
- [6] Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. Dora: Weight-decomposed low-rank adaptation. *arXiv preprint arXiv:2402.09353*, 2024.
- [7] Armen Aghajanyan, Luke Zettlemoyer, and Sonal Gupta. Intrinsic dimensionality explains the effectiveness of language model fine-tuning, 2020.
- [8] Yu-Ang Lee, Ching-Yun Ko, Pin-Yu Chen, and Mi-Yen Yeh. Learning rate matters: Vanilla lora may suffice for llm fine-tuning. *arXiv preprint arXiv:2602.04998*, 2025.
- [9] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.
- [10] Neel Redkar. Planning vs reasoning: Ablations to test capabilities of lora layers. *arXiv preprint arXiv:2412.00029*, 2025.
- [11] Ian Tenney, Dipanjan Das, and Ellie Pavlick. Bert rediscovers the classical nlp pipeline, 2019.
- [12] 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 David Yarowsky, Timothy Baldwin, Anna Korhonen, Karen Livescu, and Steven Bethard, editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA, October 2013. Association for Computational Linguistics.
- [13] John Schulman. Tinker and LoRA without regret. Guest lecture, Stanford CS224N, March 2026. Thinking Machines Lab.
- [14] John Schulman and Thinking Machines Lab. Lora without regret. *Thinking Machines Lab: Connectionism*, 2025. <https://thinkingmachines.ai/blog/lora/>.