

Building GPT-2 with Finetuning Optimizations

Stanford CS224N Default Project

Ethan Lee

Department of Computer Science
Stanford University
eth127@stanford.edu

Jingyu Zhang

Department of Chemistry
Stanford University
jluzhang@stanford.edu

Ethan Lu

Department of Math
Stanford University
ethanlu@stanford.edu

Abstract

In this project we develop a working implementation of the GPT-2 architecture from scratch in Python + PyTorch. We then use this language model, initialize it with pre-trained model weights, and use it to perform the downstream tasks of paraphrase detection and sonnet generation, aiming for accuracy of the generated results. In addition to finetuning the full model, we also implement several Parameter-Efficient Fine-Tuning (PEFT) methods, with a focus on how they interact they interact with each other. We find that our current implementation of GPT-2 successfully completes the tasks that we have tested on, and that the PEFT techniques are able to achieve similar performance with drastically sparser model updated.

1 Key Information to include

- Mentor: None
- External Collaborators (if you have any): None
- Sharing project: N/A
- Late Days:

Team member	Late days remaining
Ethan Lee	2
Jingyu Zhang	5
Ethan Lu	5
Total pooled	12
Team size	3
Late days applied for the report	3

2 Introduction

Recent advances in Large Language Model (LLM) training and finetuning techniques can be used to improve downstream task performance for older foundation models, specifically the GPT-2 family of models released in 2019 [1]. The shift from model architecture to computational efficiency means that it becomes meaningful to pay closer attention to what models are doing at a lower level. Improving smaller models like GPT-2 is not only valuable from an academia perspective but also a necessity for usage in environments where memory and GPU availability are strictly constrained during the training/testing process.

The difficulty in constructing these models lies in the trade-off between performance and flexibility: full-parameter fine-tuning is often computationally costly, while simple linear approaches often fail to capture all the nuances in downstream tasks. To address this, we explore the power of Parameter-Efficient Fine-Tuning (PEFT).

In this project, we investigate a few techniques to narrow the performance gap between older models and contemporary methods:

1. **Low-Rank Adaptation (LoRA)** [2]: updating only low-rank decompositions of weight matrices to reduce the trainable parameter count.
2. **Bottleneck adapters**: inserting specialized modules between transformer layers to capture the most relevant task-specific features.
3. **Prefix tuning**: train vectors specific to the downstream task to be prepended to the input of every layer.

By benchmarking these approaches on the tasks of sentiment detection, paraphrase detection, and sonnet generation, we aim to answer the following questions:

- Can we quantify the performance characteristics of only doing low rank updates? Since GPT-2 is an older and hence relatively smaller model, a comparison benchmark against full rank updates is actually feasible, allowing us to do such experiments locally.
- How well do these changes compound upon each other? Can distillation still be effective when restricted to low rank updates, or are full weight updates typically necessary to realize the full impact?

While PEFT has grown as promising solutions to the aforementioned issue, it is not yet clear how the methods included in this category interact with each other when combined. Existing research has benchmarked different PEFT approaches separately, but a clear experiment on their combined efficacy has yet to be performed. Stacking multiple PEFT techniques could be beneficial, but it has yet to be shown whether this is actually the case and if there is a specific subset of these techniques that optimizes model performance.

3 Related Work

Our PEFT approaches were based on the works of [2], [3], and [4].

Low-Rank Adaptation (LoRA) [2] is based on the insight that the change in weights during model adaptation has a low "intrinsic rank." By constraining updates to a low-rank decomposition of the original weight matrices, LoRA reduces the number of trainable parameters while keeping performance competitive. We incorporate LoRa to its efficacy when the rank r is restricted to 16 on the GPT-2 architecture.

On the other hand, prefix tuning adds task-specific tokens to the hidden states of every layer in a transformer. [3] This provides persistent context to point the frozen model's attention towards the task at hand. While this paper demonstrated success in generative tasks like table-to-text, our work evaluates focuses on the efficacy of this approach in paraphrase detection, where we observe the effect of using a fixed prefix length of 64.

Finally, bottleneck adapters involve the addition of trainable feed-forward modules between the layers of a frozen transformer [4]. These modules project the hidden state into a lower-dimensional (64 in this case) bottleneck before expanding it back up to the original number, allowing the model to acquire task-specific information with minimal overhead.

4 Approach

Our approach for building the language model to complete the downstream tasks is as follows:

- We first implement the GPT-2 architecture as specified in the project specifications, using it as a language model that predicts the next word of any given context.
- Namely, this uses:
 - masked multiheaded self attention (as described in the "Attention is All You Need" paper [5]):

$$\text{Attention}(Q, K, V) = \text{Softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V$$

$$\text{MultiHead}(Q, K, V) = \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O$$

where

$$\text{head}_i = \text{Mask}(\text{Attention}(QW_i^Q, KW_i^K, VW_i^V)),$$

where Mask masks all entries strictly above the diagonal by setting them to $-\infty$

- position-wise feed-forward networks with ReLU activation functions:

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2,$$

and

- dropout after each attention layer and MLP with

$$p_{\text{drop}} = 0.1.$$

- The inputs to the model are tokenized and embedded before being processed by the model, taking the embedding of the last token, and unembedding it to return the next predicted word.
- We wrote an Adam optimizer to aid in training the model, avoiding issues such as instability during training by calculating adaptive learning rates for the model with its first- and second-order moments.
- For paraphrase detection, as a baseline to compare our model to, we calculated the pre-finetuning, raw likelihood of the words "yes" and "no" appearing at the end of a prompt, without any training on the Quora dataset. The accuracy was 39%.

We now describe implementation details of the PEFT techniques we evaluated:

4.1 LoRA implementation

The first PEFT method we chose to explore was low rank adaption, as described in [2]. In this method, for each attention layer, we add a total of four trainable rank 16 matrices: $A^Q, A^V \in \mathbb{R}^{16 \times 768}$ and $B^Q, B^V \in \mathbb{R}^{768 \times 16}$. We then initialize $B^Q = B^V = 0$, and sample A^Q, A^V from a Kaiming uniform distribution with parameter $\sqrt{5}$. We then update

$$W^Q \leftarrow W^Q + B^Q A^Q, W^V \leftarrow W^V + B^V A^V$$

and freeze all the original GPT-2 parameters.

4.2 Adapters implementation

To avoid training all parameters of a model for each new task that the model is given, we decided to implement bottleneck adapters, which were proposed and described in detail by Houlsby et al [4]. In their paper, this strategy allowed the model, after using only 3% of all parameters, to perform similarly to finetuning using all parameters.

In our case, for each of the GPT-2 transformer layers (12 total), we added an adapter block after the multi-headed self-attention sublayer, as well as after the feed-forward sublayer. We have a down-projection layer that decreases the number of hidden dimensions from 768 to 64. We also have a subsequent layer that projects the bottleneck back to the original size. We could have fixed the adapter size to 8 or 256 as well. Selecting the best bottleneck size could be a good starting point for further work on this extension.

4.3 Prefix Tuning implementation

As described in [3], for prefix tuning, we prepended 64 trainable virtual key/value embeddings to each attention layer, with each embedding sampled from a standard Gaussian.

During training for all of the above methods, we froze the original GPT-2 parameters. Only the parameters relevant to the methods above were allowed to update.

5 Experiments

The experiments that we used to test the model’s performance on various downstream tasks is as follows:

5.1 Data

5.1.1 Sentiment analysis

We use the Stanford Sentiment Treebank and CFIMDB datasets, which consist of several thousand movie reviews each with a determined sentiment labeling provided by human judges. We employed a train-dev-test split of approximately 70-10-20.

5.1.2 Paraphrase detection

We use the Quora dataset, containing 400,000 question-pairs as well as whether each pair contains two questions which are paraphrases of each other. Analyzing the dataset yields that in order to be considered a paraphrase, the two questions must ask the same question down to the detail; any inclusion or exclusion will cause the pair to be marked as not a paraphrasing. We employed a train-dev-test split of approximately 70-10-20.

5.1.3 Sonnet generation

We used 154 sonnets written by Shakespeare. We used 143 sonnets as train and 12 as test data. These were used as an instance of supervised learning, treating the first three lines of each sonnet as the input data, and the remaining lines of the sonnet as the true output data. Performance was measured by the chrF score of the generated sonnets as compared to the true sonnets.

5.2 Evaluation method

For sentiment analysis, the metric we used to determine performance was accuracy and F1 score; accuracy was used as the main metric for judging performance, but we also calculated F1 score to give additional context into the model’s performance. For sonnets, we used the thing from the handout.

5.3 Experimental details

Training was performed on an NVIDIA H100 GPU via the Modal Cloud computing platform for accelerated computation. Our hyperparameters for the AdamW optimizer were as follows:

Table 1: Hyperparameter configurations across tasks. All tasks utilized the AdamW optimizer and a Cross-Entropy loss function.

Task	Learning Rate	Batch Size	Epochs
Sentiment Analysis (Full Finetuning)	1×10^{-5}	16	10
Sentiment Analysis (Last Linear Layer)	1×10^{-3}	8	10
Paraphrase Detection	1×10^{-5}	64	10
Paraphrase Detection (PEFT Evals)	1×10^{-5}	32	5
Sonnet Generation	5×10^{-5}	64	20

All parameters for the PEFT configurations were set as described in section 4.

- **Paraphrase detection:** Our inputs for paraphrase detection were structured using the default prompt format: "Is s1 a paraphrase of s2? Answer yes or no:"
- **Sonnet generation:** A couple of avenues were explored to increase the authenticity of the generated sonnets:
 - The baseline involved finetuning all layers, which may have led to disturbances across layers when trying to fit the model to the sonnets. Multiple combinations of freed layers were tried, and the one that produced the best results was by freezing all layers except the last four out of twelve transformer layers as well as the final norm layer.

- A beam search method was attempted which maintained the five most probable lines, based off of log-likelihood, out of multiple possible generated next tokens. This often caused the model to go in loops, so on top of this, a ban on repeat n -grams for $n = 3, 4$ was implemented. Still, this failed to improve over the baseline.
- To constrain the model to conform to the sonnet format, probability masking was done to ensure the sonnet stopped at exactly 14 lines, no double newlines were added, and only the last two lines started with two spaces. This improved the formatting of the generated sonnets to match more closely the true sonnets.
- A more advanced prompting technique was also attempted. The model would start by generating the entire sonnet as normal, then would be prompted to give a summary of the contents of the sonnet. The model would then be reprompted with “This is a sonnet about <description>”, then be given the first three lines, then generate the sonnet again.

5.4 Results

5.4.1 Sentiment analysis

This resulted in the following accuracies for the corresponding datasets for sentiment analysis and finetuning methods below:

Dataset	Finetuning Method	Accuracy
SST Dev	Last-Linear	0.464
	Full Model	0.506
SST Test	Last-Linear	0.475
	Full Model	0.547
CFIMDB Dev	Last-Linear	0.861
	Full Model	0.984

We submitted the full-model results to the leaderboard.

5.4.2 Paraphrase detection

The table below details the increase in accuracy and decrease in loss during training across the 10 epochs with full parameter tuning. The accuracy achieved at the end was 89.6%; we submitted this run to the leaderboard.

Epoch	Loss	Dev Accuracy
0	0.411	85.3%
1	0.317	86.4%
2	0.280	87.9%
3	0.252	87.8%
4	0.228	88.6%
5	0.206	88.7%
6	0.186	89.2%
7	0.167	89.2%
8	0.150	89.4%
9	0.134	89.6%

In addition, below are the best accuracies achieved using the test data, which were submitted to Gradescope.

Method	Accuracy
Full Parameter Tuning	86.2%
Adapters	80.0 %
Adapters + LoRA	75.7%

We also ran experiments with all 8 combinations of the different PEFT methods. Although we also wanted to include wall clock training time for each of these approaches, due to high variations in

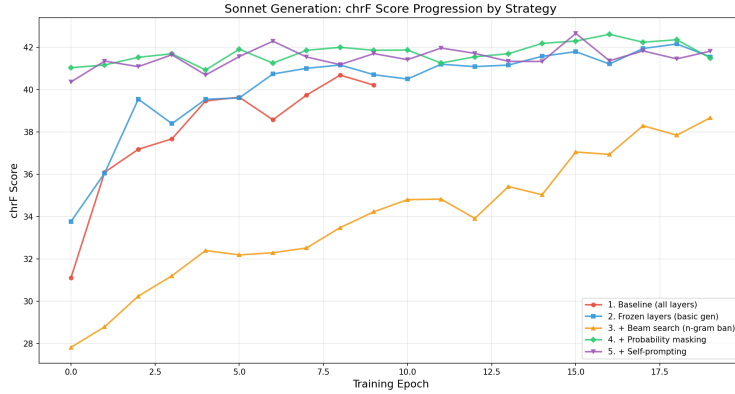
provider compute efficiency (e.g. the same parameters doubling in runtime) and limited compute time, we were unable to add that to this comparison.

Table 2: Performance metrics for combinations of PEFT methods after training for 5 epochs

Method	Training Loss	Dev Accuracy	Trainable Parameters
<i>Baseline</i>			
Full Parameter Tuning	0.175	0.891	125.03 M
<i>Single Methods</i>			
LoRA	0.412	0.811	591 K
Prefix Tuning	0.628	0.659	1.181 M
Adapters	0.319	0.863	2.380 M
<i>Paired Combinations</i>			
LoRA + Prefix Tuning	0.570	0.715	1.771 M
LoRA + Adapters	0.320	0.858	2.970 M
Prefix Tuning + Adapters	0.423	0.807	3.560 M
<i>All methods</i>			
LoRA + Prefix Tuning + Adapters	0.406	0.811	4.150 M

5.4.3 Sonnet generation

The results produced by the strategies discussed above are as follows:



Strategy	Best chrF	Best Epoch	Δ vs Baseline
1. Baseline (all layers)	40.68	8	—
2. Frozen layers (basic gen)	42.15	18	+1.47 $\uparrow$
3. + Beam search (n-gram ban)	38.67	19	-2.01 $\downarrow$
4. + Probability masking	42.61	16	+1.93 $\uparrow$
5. + Self-prompting	42.65	15	+1.97 $\uparrow$

Table 3: Performance comparison of different generation strategies.

We kept the best-performing epoch for inference.

6 Analysis

We first make some remarks regarding PEFT performance on the paraphrase detection task. LoRA and adapters generally do very competitively, but prefix tuning does not. This might be because of this method being sensitive to initialization, as noted in [3]. Although the authors of this paper suggest a different parameterization of the prefix weights to mitigate this issue, we were unable to add this to our implementation due to compute constraints. For paraphrase detection, we observe that

decrease in loss generally is typically accompanied by dev accuracy increase. While fine-tuning with all parameters gave the highest dev accuracy (table 2), the method with just adapters reached 86.3 accuracy while training on only approximately 2% of the total parameters. If we had more time, we could have increased the number of epochs from 5 to 10 (default) since dev accuracy was still increasing from epoch 4 to 5.

Upon closer examination of the results for the validation set (Table 3), we have several interesting observations.

Table 4: Randomly chosen error cases from the adapter-tuned model on the Quora paraphrase validation set.

#	Question Pair (Sentence 1 / Sentence 2)	Truth	Prediction
1	S1: Why computer vision is hard? S2: Why computer vision is computationally hard?	0	1
2	S1: Can hamsters eat cheese? S2: Can hamsters eat raisins? If so, how harmful are they for them in any way?	1	0
3	S1: Who are the people with the most followers on Quora? S2: Who is the most popular writer on Quora?	1	0
4	S1: Why still its not possible to invent a time machine? S2: Can we create a time machine?	1	0

For the false positives case, the model seems to rely word overlap even when, semantically, the two questions are different. For example, Example #1 shows that the model is overlooking the difference between "hard" and "computationally hard." Similarly, when the exact wording of the question changes significantly, the model neglects to flag duplicates despite identical meaning. For instance, Example #3 shows that perhaps the adapter layers did not learn substantially the field-specific synonyms "most followers" and "popular." Interestingly, Example #2 and Example #4 show that there are certain question pairs where it might be ambiguous even to a human person whether the second question is a duplicate of the first. For example, raisins and cheese, strictly speaking, are not the same thing. However, both questions ask about hamsters' diet, which could be what led the human judge(s) to set these as duplicate questions.

As seen in the last section of Table 1, combining multiple PEFT methods, specifically LoRA + adapters + prefix tuning, did not seem to be beneficial (accuracy 81.1%) and yielded the same dev accuracy as LoRA alone. This could have been due to conflicts caused in the gradient updates since each PEFT method might be attempting to change the model's hidden states in conflicting and thus counterproductive directions.

For sonnet generation, we analyze the different aspects of each of the modifications tried and why they may have improved or worsened the result:

- Beam search was the most significant change to the generation process, but it significantly worsened the result (lowering it from the 40s to the mid-30s). Upon analyzing the model outputs, it seemed that selecting only the top few branches out of all of the possible generated tokens, and weighting based on log-likelihood, significantly pressured the model to only pick highly likely words and not be creative. Because of this, the vocabulary used was more likely to be repetitive and not match to the true sonnets' vocab with the chrF score. Ultimately we decided to work on improving an implementation closer to the baseline approach.
- Finetuning all layers instead of just the last few layers, as implemented in the baseline, may have caused the model to "forget" its original training faster; it may have been quicker to produce sonnet-looking text but suffered from its original training data being overwritten, leading to a decrease in quality. Freezing all but the last few transformer layers may have helped with this regard, but because less parameters were being trained, the model trained slower. To aid this, we increased the learning rate and the number of epochs, producing a sizeable effect to increase the score to above 42.

- While constraining the model to produce sonnets in the required format with respect to whitespace helped in the earlier epochs when the model hadn't learned the formatting properly, the model was able to produce the required format after more epochs of training. Thus, this restriction did not really help on later epochs when the model was being evaluated, but still provided little benefit (likely due to rare cases when the model diverged heavily from the expected format).
- The prompting technique helped the model produce more centered sonnets on topic based off of manual review. The model was not trained on data which involved summarization and generating with respect to a topic, though, so it depended on its prior training in order to complete these tasks. The benefits and drawbacks may have cancelled each other out, causing this to not have that profound of an impact on the score.

7 Conclusion

The experiments that we carried out successfully implemented the GPT-2 architecture, and improved the GPT-2 model's performance over the tested tasks. In additional research, we could explore more diverse combinations of strategies for improving model performance, and exploring the use of additional hyperparameters for the PEFT layers such as varying the rank. A more exhaustive grid search for some of the hyperparameters could yield better configurations that we have not uncovered.

Team contributions (Required for multi-person team)

1. Ethan Lee: Sonnet generation implementation.
2. Ethan Lu: Base model + Adam implementation, sentiment classification, LoRA implementation, prefix tuning implementation.
3. Jingyu Zhang: Paraphrase detection, adapter implementation.

References

- [1] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.
- [2] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- [3] Xiang Lisa Li and Percy Liang. Prefix-tuning: Optimizing continuous prompts for generation. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4582–4597, Online, August 2021. Association for Computational Linguistics.
- [4] Neil Houlsby, Andrei Giurgiu, Stanislaw Jastrzebski, Bruna Morrone, Quentin De Laroussilhe, Andrea Gesmundo, Mona Attariyan, and Sylvain Gelly. Parameter-efficient transfer learning for NLP. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR, 09–15 Jun 2019.
- [5] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023.