

Building GPT-2

Stanford CS224N Default Project

Suzannah Wistreich

Department of Computer Science
Stanford University
suzannah@stanford.edu

Abstract

Past NLP systems relied on explicit supervised learning with datasets and architectures varying by task. These systems showed promising performance on narrow benchmarking, however, consistently fail to generalize broadly. This project implements GPT-2 from scratch, and using pretrained weights evaluates performance on a suite of downstream tasks. With this, we aim to investigate if token prediction alone provides sufficient language representations that can transfer across diverse tasks without needing task-specific datasets or architectures.

1 Key Information

- Mentor: N/A
- External Collaborators (if you have any): N/A
- Sharing project: No
- Late Days (if you are using any):

Team member	Late days remaining
Suzannah Wistreich	0
Total pooled	0
Team size	1
Late days applied for the report	3

2 Introduction

Past NLP systems relied on explicit supervised learning with datasets and architectures varying by task. These systems showed promising performance on narrow benchmarking, however, consistently fail to generalize broadly. This project implements GPT-2 from scratch, and using pretrained weights evaluates performance on a suite of downstream tasks. With this, we aim to investigate if token prediction alone provides sufficient language representations that can transfer across diverse tasks without needing task-specific datasets or architectures.

In this project, we implement the core components of GPT-2 from scratch, including masked multi-head attention, positional embeddings, transformer blocks, and the AdamW optimizer. We then evaluate the resulting model on three downstream tasks from the CS224N default project: sentiment analysis, paraphrase detection, and sonnet generation.

Finally, we explore parameter-efficient fine-tuning using Low-Rank Adaptation (LoRA), which freezes the pretrained model weights and learns low-rank updates to adapt the model to new tasks with far fewer trainable parameters. Through these experiments, we analyze how model size, fine-tuning strategy, and parameter-efficient adaptation affect downstream performance.

3 Related Work

In recent years, transformer-based language models have become the predominant architecture for natural language processing. Introduced in *Attention is All You Need* by Vaswani et al. [1], the transformer architecture illustrated that self-attention alone is sufficient to model long-range dependencies in input data. This property is particularly useful for NLP, where written text meaning relies on long-range sequence dependencies, replacing previous convolution or recurrent methods.

Stemming from this, GPT-2 was introduced in 2019 by Radford et al. [2], a large autoregressive language model trained via text-token prediction on a large general text corpus sourced from the internet. While previous NLP systems relied on explicit supervised learning with datasets and architectures varying by task, GPT-2 demonstrated marked generalization capabilities, performing well across a wide suite of downstream tasks through fine-tuning or prompting, *without* the use of per-task architectures or datasets for core training.

However, even with competitive advances in generalization, large language models can be prohibitively expensive to train and fine-tune due to their large number of parameters. Recent methods have explored parameter-efficient fine-tuning (PeFT) approaches, which adapt pretrained models using a much smaller number of trainable parameters. One popular method is Low-Rank Adaptation (LoRA) [3], which operates by first freezing the original pretrained model weights, then instead learning lower-rank updates inserted into existing weight matrices. This has the effect of drastically shrinking the number of learnable parameters during fine-tuning, often leading to comparable downstream task results. With this, large language models can still be adapted to new tasks with notably lower computation and memory requirements.

4 Approach

4.1 Main Approach

We implement *GPT-2* from scratch, a decoder-only transformer trained via next-token prediction. We first developed core general components of GPT-2, specifically masked multi-head attention, positional and token embeddings, and the forward pass for our core transformer blocks. Additionally, we developed the AdamW optimizer step with moment estimation and bias correction to aid in training. After these base components were completed, then we implemented the forward pass for the Paraphrase Detection task, using our base GPT-2 model to predict binary detection of paraphrase pairs. Model structure and scaffolding outside of these components were provided in the project handout.

4.1.1 Masked Multi-Head Attention

Masked Multi-Head Attention allows our GPT-2 model to jointly attend to data at different positions, rather than averaging representations into a single attention head in the forward pass. We compute a scaled dot-product applied across multiple attention heads.

Specifically, we define single-head attention with queries Q , keys K , and values V as:

$$Attention(Q, K, V) = Softmax\left(\frac{QK^\top}{\sqrt{d_k}}\right)V \quad (1)$$

Note that GPT-2 uses masked attention, blocking the model from attending to future positions in text data. We implement this by simply applying an upper-triangular mask to the attention weights.

4.1.2 Embeddings

Our method implements two embedding types after using byte pair encoding (BPE) tokenization: token and positional embeddings. **Token embeddings** map individual ids in an input sequence into vector representation for model use. **Positional embeddings** do not encode input content itself, but rather encode the *position* of individual words within the input. For GPT-2, positional embeddings are learned for all 1024 positions in a model input. Finally, embeddings are summed element-wise to produce the final embedding for each token position.

4.1.3 Transformer Block Architecture

We put implemented and provided components together in our transformer’s forward pass. We first apply pre-layer normalization, then masked multi-head attention from Section 4.1.1, then a residual connection with dropout. We then applied a second pre-layer normalization, then a position-wise feed-forward network, and finally another residual connection with dropout.

4.1.4 AdamW Optimizer

Finally, to assist in model training, we complete the AdamW optimizer implementation. At each step, we update EMA of the gradients and squared gradients, known as the first and second moments respectively. We then apply bias correction using the efficient formulation, update our learned parameters, and then apply decoupled weight decay after the gradient update.

4.2 Baselines

We use the default GPT-2 baseline as described in the project handout. We evaluate GPT-2 with the provided fine-tuning configuration against GPT-fine-tuned with the more expressive `gpt2-medium` model on the paraphrase detection task.

4.3 Task-Specific Finetuning

After implementing core components above, we then turn to fine-tuning GPT-2 on a variety of downstream tasks. In each case, we use our pretrained GPT-2 representation and only added the lightweight modifications to task-specific heads or decoding protocols. Below we detail methodology for each task.

4.3.1 Sentiment Analysis

We build upon the core GPT-2 implementation for sentiment classification by adding a small linear classifier head. For each query sequence, we first run GPT-2 on the tokenized sequence, identify the last non-padding token, then use that hidden state as our last sentence representation. We then apply dropout with $p = 0.3$ followed by a linear layer that maps the last token feature into sentiment logits. The number of classes varies by dataset; CFIMDB uses binary indication (2 classes) whereas SST can predict within 5 classes. We test under full fine-tuning (GPT-2 parameters are fine-tuned with the classifier head) and last linear layer tuning configurations (GPT-2 is frozen while the classifier head is trained).

4.3.2 Paraphrase Detection

For the Paraphrase Detection task, we utilize a cloze-style question formulation rather than an additional classification layer. We construct a prompt asking GPT-2 to evaluate two sentences s_1 , s_2 , of the form:

“Is $\{s_1\}$ a paraphrase of $\{s_2\}$? Answer ‘yes’ or ‘no’”

We process this tokenized sequence through GPT-2. We then take the token embeddings and use the final token’s hidden state, projecting it into vocabulary logits using GPT-2’s weight-tied output embedding matrix. This weight tying reuses the same weights used for the input token embeddings to map hidden states back into vocabulary space, allowing us to treat the prediction as next-token language modeling. From there, the model can simply assign probability to tokens like “yes” and “no.”

4.3.3 Sonnet Generation

As our primary generation task, we adapt GPT-2 to predict a vocabulary distribution at *every* token position, rather than only the last token. Following our fine-tuning method, we first process the target input sequence through GPT-2, then take the full sequence of hidden states and project them back into vocabulary logits once again using GPT-2’s weight-tied output embedding matrix. This lets the model train autoregressively with next-token prediction. See Section 5.3 for train and inference time details.

4.4 PeFT/LORA Extension

Finally, we explore parameter-efficient fine-tuning (PeFT) methods for adapting large language models to perform downstream tasks while reducing training resource required. Specifically, we explore *Low-Rank Adaptation (LoRA)* [3], which is a method that allows efficient fine-tuning by freezing the pretrained model weights while learning a small subset of task-specific parameters.

LoRA modifies a pretrained weight matrix $W \in \mathbb{R}^{d \times k}$ by introducing a low-rank update rather than directly updating the full matrix. Instead of learning ΔW directly, LoRA parameterizes the update as the product of two low-rank matrices:

$$W' = W + \Delta W, \quad \Delta W = BA$$

where $A \in \mathbb{R}^{r \times k}$ and $B \in \mathbb{R}^{d \times r}$, with $r \ll \min(d, k)$. During training, the original weights W remain frozen and only the low-rank matrices A and B are optimized.

This approach drastically reduces the number of trainable parameters while preserving most of the representational capacity of the model. In our experiments, LoRA reduces the number of trainable parameters from 125M to approximately 149K (about 0.12% of the model), enabling efficient adaptation while maintaining reasonable downstream performance.

5 Experiments

5.1 Data

We use the datasets provided in the default project, detailed below:

- **Stanford Sentiment Treebank (SST) dataset:** 11,855 single sentences from movie reviews, rendering 215,154 unique phrases each annotated by 3 human judges. (*Sentiment Analysis*)
- **CFIMDB dataset:** 2,434 highly polar movie reviews, possibly longer than a single sentence. (*Sentiment Analysis*)
- **Quora Dataset [4]:** large subset of 400,000 question pairs with labels indicating whether particular instances are paraphrases of one another. (*Paraphrase Detection*)
- **Sonnet Dataset:** 154 sonnets written by Shakespeare. (*Sonnet Generation*)

We do not use additional external data, and minimal preprocessing is handled for us in the default project dataset architecture.

5.2 Evaluation method

For each task, we utilize the evaluation metric specified in the default final project handout. Specifically, we use the following metrics:

- **Sentiment Analysis:** Classification accuracy on SST and CFIMDB, comparing our GPT-2 models against the provided baseline configurations stated above.
- **Paraphrase Detection:** Classification accuracy on the Quora dataset, comparing our base GPT-2 model against the task-specific fine-tuned variant. Predictions are obtained from GPT-2’s next-token distribution by comparing the probabilities assigned to the tokens “yes” and “no”, and selecting the higher-probability token, as described in Section 4.3.2.
- **Sonnet Generation:** chrF score, measuring character-level n -gram overlap between generated and ground-truth sonnet continuations. Additionally, we qualitatively inspect the generated poems for reasonable fluency and coherence.

5.3 Experimental details

For all experiments, we implement our work in PyTorch and run training and inference on a single NVIDIA L40S GPU. We use our GPT-2 implementation initialized with pretrained weights from HuggingFace. For training, we use the AdamW optimizer described in Section 4.1.4, with $\beta_1 = 0.9$,

$\beta_2 = 0.999$, $\epsilon = 10^{-6}$, and bias correction. We also fixed the randomization seed to 11711 for all stochastic components (ex., Python, NumPy, PyTorch). For downstream tasks, we utilize the following details:

- **Sentiment Analysis:** We train GPT-2 classification heads on the SST and CFIMDB datasets described above, using the GPT-2 tokenizer and EOS token as padding. We gather the final non-padding token representation, followed by dropout and a linear layer as described in Section 4.3.1
- **Paraphrase Detection:** We format paraphrase detection as a cloze-style generation task, where GPT-2 predicts the next token corresponding to “yes” or “no” after a prompt containing the two candidate sentences s_1, s_2 . We train with learning rate 10^{-5} and batch size 8, and with checkpoints selected using development-set accuracy.
 - *LoRA Extension:* For our extension, we insert low-rank modifications into our attention projections, freezing the rest of the GPT-2 weights. This results in 148,994 trainable parameters out of 125,179,394 total parameters.
- **Sonnet Generation:** We fine-tune GPT-2 on sonnets using standard next-token prediction with cross-entropy loss. We use a learning rate of 10^{-5} and batch size 8. During generation, we condition on the first three lines of each held-out sonnet, and generate the continuation autoregressively using nucleus sampling with top $p = 0.9$.

5.4 Results

With the experimental details above, below we share our results and analysis for our GPT-2 experiments. We submitted the GPT-2 small configuration to the leaderboard following the default project setup. In our experiments, we also evaluated GPT-2 medium, which achieved slightly higher accuracy. Results for Paraphrase Detection and Sonnet Generation are tabled in Table 1.

Model	Para Dev (%)	Para Test (%)	Sonnet Dev (chrF)	Sonnet Test (chrF)
GPT-2 (small, baseline)	87.9	86.3	41.70	40.99
GPT-2 Medium	89.6	88.2	42.03	40.8

Table 1: Performance of our GPT-2 models across tasks. We report development and test results for paraphrase detection and sonnet generation.

We table our results for Sentiment Analysis in Table 2.

Model	Baseline Dev Acc (%)	Our Dev Acc (%)	Our Test Acc (%)
<i>SST</i>			
GPT-2 (Last Linear Layer)	46.2	46.6	45.1
GPT-2 (Full Fine-Tuning)	51.3	51.6	54.3
<i>CFIMDB</i>			
GPT-2 (Last Linear Layer)	86.1	89.0	
GPT-2 (Full Fine-Tuning)	97.6	98.0	

Table 2: Sentiment classification accuracy on SST and CFIMDB. We compare our results with the baseline development accuracies provided in the project handout.

Finally, we present results for the LoRA/PeFT extension in Table 3.

6 Analysis

6.1 Overall Results and Capacity Experiment

Across these tasks, our results demonstrate that increasing model capacity modestly improves downstream performance. As shown in Table 1, GPT-2 medium achieves higher paraphrase detection

Model	Trainable Params	Total Params	Para Dev Acc (%)
GPT-2 Full Fine-Tuning	125M	125M	87.9
GPT-2 + LoRA (r=4)	149K	125M	80.5

Table 3: Comparison between full fine-tuning and LoRA-based parameter-efficient fine-tuning on paraphrase detection. LoRA significantly reduces the number of trainable parameters by learning low-rank updates while keeping the pretrained GPT-2 weights frozen.

accuracy than GPT-2 small, improving development-set accuracy from 87.9% to 89.6% and test accuracy from 86.3% to 88.2%. This suggests that the larger model is able to capture richer contextual representations useful for semantic similarity tasks. For sonnet generation, however, the performance is virtually the same: GPT-2 medium slightly increases dev chrF from 41.70 to 42.03, while test chrF remains similar.

6.2 Sentiment Analysis

For sentiment analysis, we observe that full fine-tuning consistently outperforms last-layer tuning. On SST, full fine-tuning improves test accuracy from 45.1% to 54.3%, demonstrating that allowing the model to adapt *internal* representations as well is beneficial for tasks requiring nuanced sentiment understanding. On CFIMDB, both configurations perform strongly, though full fine-tuning still achieves slightly higher dev accuracy.

6.3 LoRA Analysis

LoRA extension illustrates the trade-off between parameter efficiency and performance. As shown in Table 3, LoRA reduces the number of trainable parameters from 125M to only 149K while maintaining reasonable performance, achieving 80.5% dev accuracy on paraphrase detection. Although this is lower than the 87.9% achieved by full fine-tuning, it demonstrates that low-rank adaptation can retain much of the model’s capability while dramatically reducing the number of parameters that must be trained. We utilize only 0.119% of the parameters from the original experimental configuration.

7 Conclusion

In this project, we implemented GPT-2 from scratch and evaluated it across sentiment analysis, paraphrase detection, and sonnet generation tasks. Our results show that larger models improve performance on classification tasks such as paraphrase detection, while full fine-tuning generally outperforms last-layer tuning for sentiment analysis.

We also explored parameter-efficient fine-tuning using LoRA, which dramatically reduces the number of trainable parameters while maintaining reasonable performance. Although LoRA performs slightly worse than full fine-tuning, it demonstrates that large language models can be adapted effectively with far fewer trainable parameters.

References

- [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*, 2017.
- [2] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. Technical report, OpenAI, 2019.
- [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. *arXiv preprint arXiv:2106.09685*, 2021.
- [4] Shankar Iyer, Nikhil Dandekar, and Kornél Csernai. First quora dataset release: Question pairs, 2018.