Fine-Tuning on Pre-Training Data: A Model Comparison

Evan Gronberg

August 23, 2024

Table of Contents

Introduction

The goal of this paper is to determine the effect of fine-tuning a model on data that it has already been pre-trained on. To accomplish this, we take a small pre-trained model and use two different methods, standard fine-tuning and parameter-efficient fine-tuning (PEFT), to fine-tune the model on the full text of the Bible, which is included in major pre-training datasets. From there, we leverage a biblical question-answer dataset to evaluate the performance of these three model versions. Hypothetically, fine-tuning will result in better performance than the base pre-trained model displays, with PEFT resulting in less improvement than standard fine-tuning. As the results section details, however, this hypothesis is decidedly wrong in this instance, with the fine-tuned models displaying only about half the question-answering accuracy of the base model.

Background

LLMs are often trained in multiple phases. The first of these phases is known as pre-training. Pre-training takes a completely untrained LLM and shows it a large corpus of textual data from which it infers linguistic patterns. This phase is almost

always the one that leverages the most data; as an example, the recently released Llama 3.1 models were trained on “over 15 trillion tokens.” [1]

From there, other types of training can be applied to the same model, perhaps the most well-known of which is fine-tuning. There are multiple ways to fine-tune, but the most straightforward way of doing so is to simply show the pre-trained model more data. This is a compute-intensive process since all the model’s parameters are involved, but there is a second way of accomplishing this with less compute: parameter-efficient fine-tuning (PEFT).

There are multiple ways of accomplishing PEFT, but one of the best-known ways is low-rank adaptation (LoRA). This method leverages the ability to represent a matrix as a multiplication of smaller matrices, and it fine-tunes those smaller matrices instead. For example, a 100×100 matrix can be decomposed into two matrices of size 100×2 and of size 2×100, which when multiplied together produce a 100×100 matrix; we thus reduce a 10,000 parameter matrix to a 400-parameter representation.

Methods

Our approach to answering this question starts where any such question must: with data. The Bible was chosen for this fine-tuning task for two primary reasons: (1) it is a large corpus of text at around 850K words and (2) it is in the public domain (the American Standard Version was selected for this task). [2] To load in this data for LLM training, the `datasets` Python package produced by HuggingFace is used. [3] Furthermore, due to its cultural prevalence and the wealth of information it contains, the Bible unsurprisingly has an associated question-answer dataset (BibleQA), which will be used for model evaluation. [4]

From there, a model must be selected. Due to limited time and resources, a model that is small compared to today’s most popular models needed to be selected. The GPT-2 family of models was selected for testing (see the “Results” section for commentary on specific model selection). To retrieve this model, as well as its associated tokenizer, the `transformers` Python package produced by HuggingFace is used. [5]

The `transformers` package also provides basic fine-tuning capability (since basic fine-tuning is simply a matter of more training), but for PEFT, we leverage the aptly named `peft` Python package. [6]

Results

Computing Environment

Google Colab was used to run all experiments associated with this project. A runtime with an A100 GPU was selected, which had the following specifications:

- System RAM: 83.5 GB
- GPU RAM: 40.0 GB
- Disk Space: 201.2 GB

Model Selection

The GPT-2 family of models is hosted on Hugging Face at OpenAI's community page. [7] It comes in four different varieties, which are detailed in the table below.

Model Name	Parameter Count	No. of Downloads (at time of writing)
`gpt2`	137M	7.02M
`gpt2-medium`	380M	297K
`gpt2-large`	812M	750K
`gpt2-xl`	1.61B	147K

All of these models are of the text generation variety (furthermore, they also all use F32 tensors).

The plain `gpt2` is actually the smallest model of the bunch, but it is still by far the most popular. Experiments were first run with this model, but its low output quality quickly became apparent. The `gpt2-xl` model was thus tested in an attempt to compensate. This model, however, proved too cumbersome for fine-tuning on the 40 GB of GPU RAM available, so the `gpt2-large` model was selected after successful test against the GPU RAM constraint. The `gpt2-large` model was observed consuming GPU RAM in the low 30s during standard fine-tuning and the mid 30s during PEFT. The increased GPU RAM use of PEFT is explained by the additional, low-rank matrices that are added for fine-tuning.

Fine-Tuning Process

As discussed above, the original `gpt2-large` model was fine-tuned using two different methods: standard fine-tuning and PEFT. Both of these methods were

run for 4 epochs (i.e., the model was fine-tuned on the Bible for 4 complete passes), and the batch size for both runs was 8 (the next multiple of two, 16, was too large for GPU memory).

For PEFT, the LoRA was implemented. A rank (i.e., attention dimension) of 8 was used, which is simply the default when using the `peft` package's `LoraConfig` object. What required more attention, however, was the selection of target modules. The `gpt2-large` model features the following structure:

```
GPT2LMHeadModel(
  (transformer): GPT2Model(
    (wte): Embedding(50257, 1280)
    (wpe): Embedding(1024, 1280)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-35): 36 x GPT2Block(
        (ln_1): LayerNorm((1280,), eps=1e-05, elementwise_affine
        (attn): GPT2SdpaAttention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((1280,), eps=1e-05, elementwise_affine
        (mlp): GPT2MLP(
          (c_fc): Conv1D()
          (c_proj): Conv1D()
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (ln_f): LayerNorm((1280,), eps=1e-05, elementwise_affine=Tru
  )
  (lm_head): Linear(in_features=1280, out_features=50257, bias=F
)
```

The bulk of the model is a series of 36 `GPT2Block` modules. The parameter-containing modules within a given `GPT2Block` are `c_attn` (convolutional attention), `c_fc` (fully connected convolutional), `c_proj` (convolutional projection). (`c_proj` being a module in both the attention portion and the multi-layer perceptron (MLP) portion.) The parameter counts for a given `GPT2Block` are detailed in the table below.

Layer Name	Parameter Count
`c_attn`	4.92M
`c_proj` (in the attention portion)	1.64M
`c_fc`	6.56M
`c_proj` (in the MLP portion)	6.55M

These all contain a very high number of parameters, so LoRA was applied to all of them.

Figure 1 displays the loss that resulted from the standard fine-tuning process. Note the 4-step pattern that seems to have resulted from the 4 epochs run.

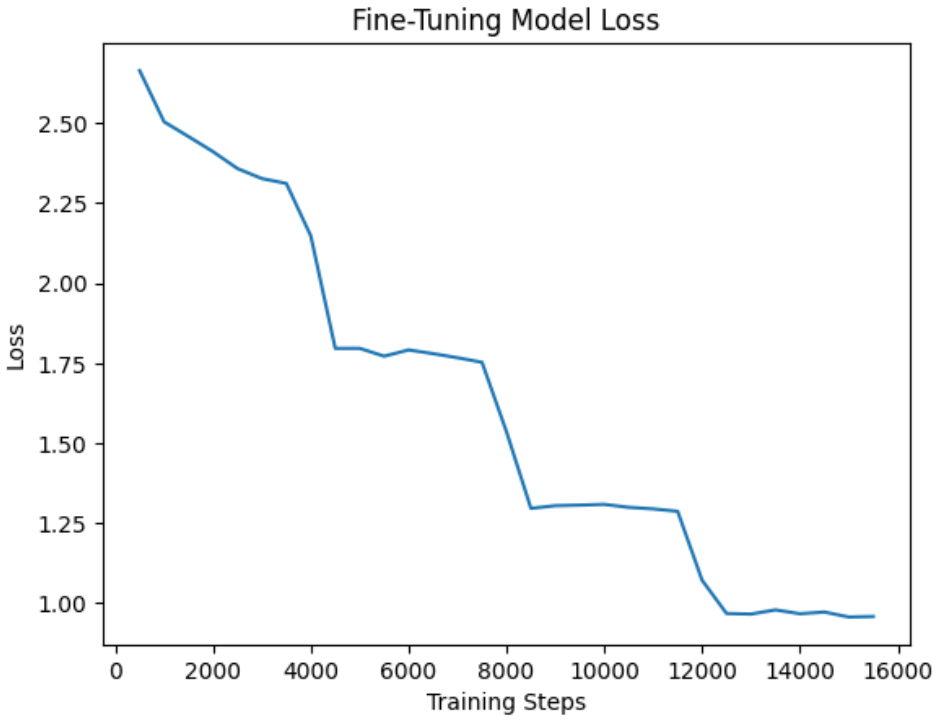

Figure 1: Model loss on standard fine-tuning.

PEFT, however, does not display this step-like pattern in its loss (see Figure 2). Unlike standard fine-tuning, PEFT experiences the majority of its loss reduction after 1 epoch (i.e., at around 4000 steps). This seems like a positive; however, the loss that PEFT quickly converges to is ~2.3, whereas standard fine-tuning results in noticeable loss reduction with each epoch, getting down to ~0.95 in the fourth and final epoch. It is worth noting that this means standard fine-tuning takes substantially longer to converge, but that its point of convergence is substantially lower than that of PEFT.

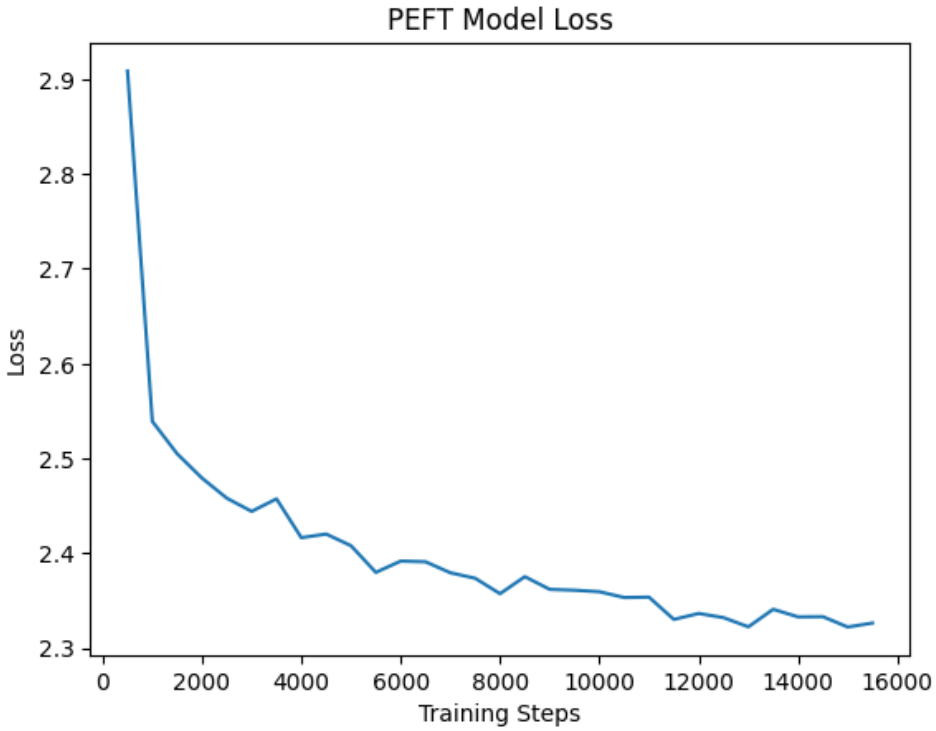

Figure 2: Model loss on PEFT.

On the A100 GPU used, the duration of fine-tuning for both methods is given in the table below.

Fine-Tuning Type	Total Duration	Duration per Epoch
Standard	5.753 hours	1.438 hours
PEFT	4.451 hours	1.113 hours

This result is, frankly, surprising. PEFT only introduces a 22.6% fine-tuning time reduction, despite the technique being applied to the majority of the model.

Model Performance

Loss is really only an approximator of relative performance; in other words, it can give us a good idea of how models can be expected to compare to one another, but it means little in terms of performance in real-world use cases.

To evaluate our fine-tuned models, we compare them to our baseline `gpt2-large` model. We do this by having each of our three models answer the set of 886 Bible-related questions provided by the BibleQA dataset discussed above.

As also discussed above, the GPT-2 family of models are text generation models, meaning that they don't exhibit a particular behavior pattern other than to produce text that continues the pattern of the prompt. Simply feeding the model a question, therefore, often results in irrelevant non-answers; in the case of the fine-tuned models, the model would tend to simply continue with biblical-sounding text. To get around this, prompt engineering was incorporated; the question-answer pattern (as well as explicit informing of biblical context) below was used to embed the question in.

> The following are questions and answers about the Bible.
>
> Q: Who was the first man?
> A: Adam
>
> Q: What was special about Samson?
> A: He was very strong
>
> Q: Where was Jesus born?
> A: Bethlehem
>
> Q: [BibleQA question here]
> A:

This engineered prompt resulted in noticeably more answer-like model responses.

Furthermore, the BibleQA dataset provides very short ground truth answers for its questions, so the max number of new tokens for the model to produce was set to

15. This ended up being a little more than necessary (model responses were always longer than ground-truth answers), but in some cases it does give the model a little more time to “think” and produce a better response.

Once each model had answered each of the BibleQA questions using the prompt format above, its answers were compared to the BibleQA’s ground-truth answers. A lenient evaluation strategy was implemented to estimate model performance; an answer was considered correct if the short ground truth answer was present anywhere in the model’s response. Simple accuracy was then recorded. Even with this leniency, however, results were poor (see the table below).

Model	Accuracy	Number Correct (Out of 886)
GPT-2 Large (Pre-Training Only)	2.37%	21
Fine-Tuned GPT-2 Large (Standard)	0.902%	8
Fine-Tuned GPT-2 Large (PEFT)	1.02%	9

For greater insight into these results, a Venn diagram of correct answers is provided in Figure 3. Given the results in the table above, it is no surprise to see the pre-trained model getting the most correct answers that neither other model got, but it is surprising to see that there was not a single instance of both fine-tuned models getting a correct answer that the pre-trained model missed. It is also worth noting that our three models combined got a total of 31 questions right, for a “mixture of models” accuracy of 3.50%.

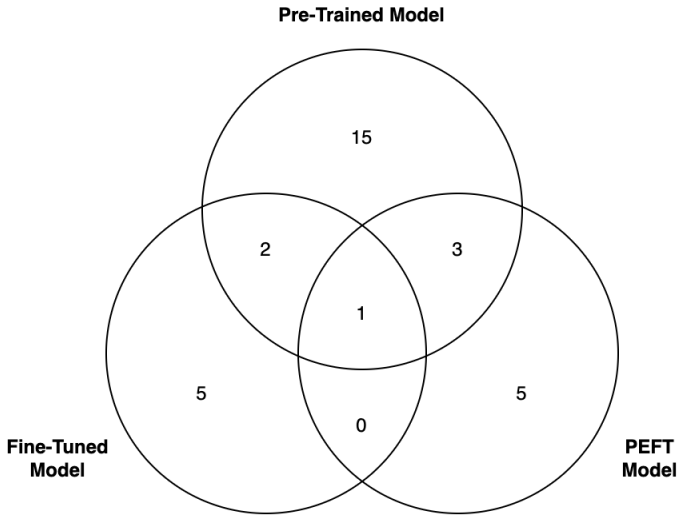

Figure 3: Venn diagram of model correct answer counts.

Our lenient evaluation criteria does mean that some actually-wrong answers are considered correct, but fortunately this effect was limited. Manual inspection of answers to the 31 questions that at least one model was evaluated to have gotten correct shows that only 5 supposedly-correct answers were actually wrong (see questions 37, 63, 100, 652, and 711 in the associated notebook). Interestingly, our evaluation criteria also allowed at least one actually-correct answer to be flagged as wrong (see question 155; the answer was “Mount Sinai,” and the PEFT model included “mount of Sinai” in its answer). All of the question-answer pairs that received a correct marking on at least one model’s response are displayed in the Jupyter notebook associated with this paper.

A fuzzy example of correctness is given in the model responses below.

Question	What was the name of Abraham's first wife?
Ground Truth Answer	Sarah
Pre-Trained Model Answer	Abraham was the father of Isaac and of Jacob.
Fine-Tuned Model Answer	Is there any more to be told of her? She was the mother of
PEFT Model Answer	Is it not Sarah? And is not Abraham the father of Sarah? And

Note that the PEFT model initially gets this question correct, but then makes a false statement (technically a rhetorical question). This reflects the LLM’s lack of reasoning capabilities; it supposes that Sarah is both Abraham’s wife and

daughter, which is not possible but reflects the simple next-token likelihood approach of the model. Also note that speaking in rhetorical questions is a fairly biblical trope, so it is interesting to see the models fine-tuned on the Bible picking up this behavior. The pre-trained model does something similar; it misses the meaning of the question completely, but it nonetheless exhibits some genealogical Bible-speak.

Finally, it is worth recording inference time for these experiments. Getting responses from each of the 3 models for each of the 886 questions took 36.18 minutes, which means that average per-inference time was 0.817 seconds.

Conclusions

It is of great interest that the *opposite* of the hypothesis stated in the introduction is correct. Fine-tuning on the Bible actually *diminished* performance on biblical questions rather than improving it. Furthermore, standard fine-tuning and PEFT resulted in near-equal performance, making standard fine-tuning seem like a potential waste of resources in some cases. That said, PEFT's time-savings were not especially notable; not to mention, it actually required *more* GPU RAM than standard fine-tuning. Thus, overall, we are left with the very important lesson that fine-tuning is not a guarantor of increased performance.

Fine-tuning is also not an easy or cheap solution to implement. Even with a GPT-2 model, now considered very small, the suite of experiments run for this project takes nearly 11 hours, which makes the speed of iterating on the solution very unwieldy under deadline pressure. Furthermore, the cost of GPU-enabled cloud compute for this small project was approximately $120.

This project was thus very educational even if not in the sense of walking away with an industry-ready approach in hand. The project was instead a highly informative deep dive into a technique that the author had had no previous experience with.

Impact and Future Work

The impact of this paper is a cautionary tale against naive fine-tuning approaches. One might easily assume that fine-tuning would somewhat easily result in greater performance within the fine-tuning domain, but this paper demonstrates

otherwise. Fine-tuning, therefore, must be carefully implemented, not as a quick and easy solution.

Future work would, perhaps somewhat obviously, seek to scale the results of this paper to larger, more modern models. (Perhaps by those with greater budget!) GPT-2 was released in 2019; in the five years since, multiple iterations of the GPT model alone have been developed. [8]

Aside from the obvious, however, future work might also include further exploration and analysis of the fine-tuning methods at play. Fine-tuning is, as this paper makes clear, not a straightforward technique, so further research would better conclude which applications fine-tuning *does* work well for rather than that for which it does not.

References

[1] Schmid, et al. “Llama 3.1 - 405B, 70B & 8B with multilinguality and long context.” Hugging Face. 2024. https://huggingface.co/blog/llama31

[2] https://openbible.com/textfiles/asv.txt

[3] Villanova, et al. “HuggingFace community-driven open-source library of datasets.” Python Package Index (PyPI). 2015. https://pypi.org/project/datasets/

[4] Zhao, et al. “Finding Answers from the Word of God: Domain Adaptation for Neural Networks in Biblical Question Answering.” arXiv. 2018. https://arxiv.org/abs/1810.12118 (for the dataset itself, see https://github.com/helen-jiahe-zhao/BibleQA)

[5] Sartran, et al. “State-of-the-art Machine Learning for JAX, PyTorch and TensorFlow.” Python Package Index (PyPI). 2016. https://pypi.org/project/transformers/

[6] Bossan, et al. “Parameter-Efficient Fine-Tuning (PEFT).” Python Package Index (PyPI). 2023. https://pypi.org/project/peft/

[7] Chaumound, et al. “OpenAI community.” Hugging Face. https://huggingface.co/openai-community

[8] Marr. “A Short History of ChatGPT: How We Got To Where We Are Today.” Forbes. 2023. https://www.forbes.com/sites/bernardmarr/2023/05/19/a-short-history-of-chatgpt-how-we-got-to-where-we-are-today/