Hadi Imtiaz

CS 387 Deep Learning

Final Project Report

LLMs: Fine Tuning, Baselines, and Character Level GPTs

Introduction

The primary objective of this project was to explore how the fine tuning of pretrained open source Large Language Models (LLMs) on two different tasks the non-fine tuned model falls behind in - Language Translation and LeetCode style Python problems. Alongside this, a character level decoder only transformer Generative Pre-trained Transformer (GPT) model was trained from scratch, to see how it compares to transfer learning.

This problem is particularly relevant today because the demand for domain specific LLMs is growing rapidly, and not every organization can afford to train or deploy models with billions of parameters from scratch. Demonstrating that a small model can be adapted for niche tasks using minimal resources is both a technical and practical contribution.

To understand this project better, we need to be familiar with a few terms:

1) Fine-tuning is the process of continuing training on a pre trained language model using a smaller, task-specific dataset. This helps the model adapt to new tasks more efficiently than training from scratch, since it already has general language knowledge.
2) Perplexity: Measures how well a model predicts the next token. A perplexity of 1.0 implies perfect prediction. The closer final perplexity is to 1, the better.
3) BLEU (bilingual evaluation understudy): Is an algorithm that evaluates the quality of machine translated text from one language to another. The score output ranges from 0 and 1 in my implementation and higher is better. A score of ~0.7 implies near perfect translation.
4) LoRA (Low-Rank Adaptation of Large Language Models): LoRA is a fine tuning technique to reduce the number of trainable parameters. Instead of updating the full weight matrices in a large model, LoRA introduces small trainable low-rank matrices into the model. These matrices contribute to changes that are relevant to the data the model is being trained on, while the base model's weights stay frozen. This is very efficient, and makes fine tuning models accessible to average consumer hardware by reducing GPU memory consumption. In this project, LoRA was applied via the peft library using rank 64 adapters during 4-bit quantized fine-tuning of LLaMA 3.2 1B
5) Quantization: A way to reduce the memory footprint and computational requirements of LLMs by converting 32 bit floating point numbers into lower precision representations, for example in this project, 4 bit quantization was implemented.
6) Decoder-Only Transformer: Simplified Neural Network architecture where only the decoder block of a transformer is used. A small decoder only transformer was implemented in this project.

Approach and Methodology:

Datasets used:
More specific information about the datasets are in included in the notebooks themselves. I used a Python LeetCode problems dataset, and two different Spanish - English datasets. I combined these datasets into one text file, and randomly assigned portions of it to a test and validation test for the Decoder only transformer lever. Hugging face's datasets library made it easier to work with datasets.

Models and Tools:
Along with Meta's LLaMa 3.2 1 Billion parameters pre trained LLM model, i used the following libraries on top of pytorch:
Unsloth: This library uses HuggingFace's transformers library to make LoRA based fine tuning faster and memory efficient.
Transformers: This library provides access to pretrained models, tokenization, and generation among other utilities.
Trl library's SFTTrainer (supervised fine tuning): This utility/library was used to manage training.
Evaluate: Used to calculate the BLEU scores.

The GPT transformer architecture was exclusively built using pytorch, however, with fine tuning, using these other libraries was essential.

Details of implementations are in the notebooks.

I decided on LLaMa 3.2 1B for a variety of reasons: It was the least computationally expensive, and the least powerful, so any changes with finetuning would be more apparent. I also felt this was more practical, because other LLaMa models would be powerful enough by themselves for most tasks.

Results and Analysis:

For the GPT trained from scratch, the output was mostly just incoherent. It reflects some sort of imitation of LeetCode-style formatting but with no logic, reasoning or consistent vocabulary. The model seems to string together words it sees together in training, however, with no coherence or semantic understanding.

However, this behavior was completely expected, and the output is not terrible considering the size of the dataset, time for training, and the limited computation power thrown at it.

Surprisingly, the perplexity score wasn't terrible. At 2.5, it implies that the model is on average unsure between 2.5 possible next tokens at each step. While far from ideal, this level of uncertainty is actually reasonable for a character-level model trained from scratch on a small dataset. It suggests that the model did manage to pick up on some structural or statistical patterns in the training data, even if it wasn't able to generate meaningful or correct solutions. Overall, the results align with expectations: the model can mimic surface patterns but lacks the depth or context to produce functional output.

As this model is mostly an educational proof of concept, calculating the BLEU scores was not a priority for me, as my model does not have the capability to handle translations.

The next model, the English - Spanish translations fine tuned LLaMa 3.2 1B, was my best performing model. However the perplexity was higher than I expected at 1.7 (still acceptable), a BLEU score from a random selection of the examples in the dataset used resulted in an output of ~0.4, which according to this source by Google means that translations are high quality. Comparing that to the base model performance, the BLEU score was so terrible that I had to double check the output predictions and their references. At ~0.146 according to the same evaluation metric, is almost useless. This is where the fine tuning really showed its practicality, even with limited training time, computational resources, and time, this produced excellent results.

The other fine tuned model, however, was disappointing. I struggled to find a way to quantitatively compare coding results, so I relied on BLEU and qualitative methods, which was just me referencing solutions online and seeing which models performed better, and from the results it seems like the base model performs better. The Fine Tuning was not a complete fail however, the results from the fine tuned model had improved formatting with no unnecessary text, just code, like from the way the model was finetuned and it mimics Python syntax more cleanly with type hints compared to the base model. However, its actual logic is obviously broken most times or completely incoherent. In many cases, it fails to solve slightly complex and some basic problems correctly, restates the prompt instead of generating code, or outputs nonsense wrapped in proper function signatures. Fine Tuning made it sound smarter without actually making it smarter, and the handful of successful generations are discounted by the other completely incoherent outputs.

This contrast between the translation model and the code generation model says a lot about how task complexity and evaluation methods affect fine-tuning success. Translation, while not trivial, is more pattern-heavy and benefits from tons of training pairs and structured sentence alignment. So even with limited resources, a small model like LLaMA 1B can actually learn useful mappings. But coding requires not just syntax, but logic, reasoning, and multi-step problem solving,things that LLaMA 1B might not have the capacity for, at least not with a small dataset and minimal training.

Conclusions, Future work

To sum everything up, I created a decoder only transformer model and trained a generative model from scratch, and then I implemented transfer learning through fine tuning and compared the results with non fine tuned models. My work was partially a success and I learned a lot about transfer learning in particular, along with making an educational model for which we went over the theoretical part in class. In particular, I learned more about how Large Language Models actually work behind the scenes in depth. I am also excited that I learned practical skills. In the current environment of the job market, fine tuning a model could actually be something I do in the future at a job/internship. Another good skill I learned from this project was related more to the tools we use as computer scientists, using a jetstream instance, installing a bunch of libraries, running into problems associated with them, a bunch of new linux commands (for example, I never used vim before), are just a few i can name off the top of my head.

Future work on this project could try to push the limits of the provided hardware, and training for a lot longer and seeing how things go from there. One thing I did learn about fine tuning is that it is very particular about the types of inputs and labels it is provided. The first time I trained my fine tuned translation model, the results were interesting. I trained it on a european parliament english - spanish translation dataset, however that made most of the outputs from the trained model output text of a similar tone. I think partly my leetcode model's performance might also have been the verbose prompts it was given, which can explain why I saw similar issues like the prompts from the training leaking into outputs for both of those instances of training. What that means for the future is that I know what kind of data finetuning is suitable for, and the next time I do something similar, I would be much better prepared. I could also redo the leetcode part of the project, but this time manually creating a suitable dataset.

In the end, this project served as both a technical and academic learning experience. With stronger data, I believe there's real potential to build something robust on top of this foundation.