Adding Information to Large Language Models

Team Member: Nicolas Ong

CMPT419 @ Simon Fraser University

Motivation and Objectives

Modern consumer LLMs like ChatGPT are used to answer questions people have. However, LLMs are unable to answer questions about information not in their training data, like recent information or domain-specific information.

For this project, our group would like to figure out what techniques are good at inserting new information into models. We believe that this is an important question as many people have information they'd like to integrate into LLMs like ChatGPT.

Approach

At a high level, here was our approach:
- Have a model that performs a task that shows it has memorized some knowledge
- Test its performance on data it wasn't taught
- Train the model to memorize this data using various techniques
- Measure how well the model memorized this data, and compare which techniques worked the best.

We decided to try 3 techniques:
- More fine-tuning
- More pre-training
- Use a KNN

In addition, we also trained the models with LoRAs (Low Rank Adapters) to see how that performed.

Data

In total, we used 4 datasets.

FEVER (Fact Extraction and VERification)
This dataset contains claims like "Anne Rice was born in New Jersey.", with labels like "SUPPORTED" or "REFUTED". This dataset is intended to be used for the FEVER task (Fact Extraction and VERification), where a model is supposed to look at provided evidence and decide if the claim is supported or refuted by the evidence. However, for the purposes of our experiments, we only used the claim and label parts of the dataset.

BH-TF (Barbenheimer True or False)
This is a custom dataset containing claims like "Barbie was released in 2023" and labels of 0 (False) or 1 (True). It was made with information that didn't exist in either GPT2's training data or in FEVER.

BH-RD (Barbenheimer Raw Data)
This is a custom dataset containing Wikipedia articles, news articles, IMDB reviews, and reddit discussions about the movies Barbie and Oppenheimer.

SW-TF (Star Wars True or False)
This is another custom dataset like BH-TF, but on data that existed when GPT2 was trained and FEVER was made.

Experimental Setup

For our base model, we used GPT2 with a classification head (gpt2-ch).

We then trained this model to answer True or False questions using the FEVER dataset. True or False is the "memorization measure" we use to see if a model is memorizing new data.

We got two base models:
- gpt2-ch-cht: only the classification head's weights were trained.
- gpt2-ch-fft: all the weights (gpt2 + ch) were trained

After this, we trained these models using the techniques we wanted to evaluate.

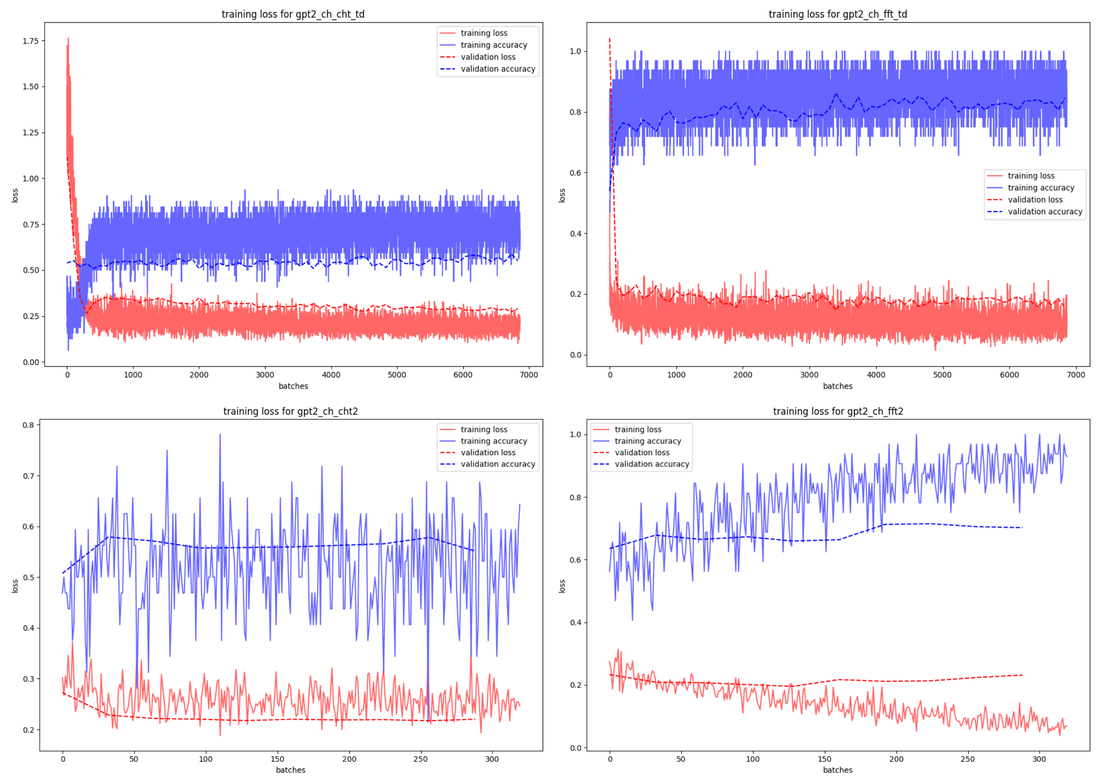
In "more fine-tuning", we further trained the model on the T/F task, but using the BH-TF dataset to add the new information into the model. We also used LoRAs here.
This got us 4 models:
- gpt2-ch-cht2: gpt2-ch-cht fine-tuned on BH-TF
- gpt2-ch-fft2: gpt2-ch-fft fine-tuned on BH-TF
- gpt2-ch-cht2q: gpt2-ch-cht fine-tuned on BH-TF, with LoRAs
- gpt2-ch-fft2q: gpt2-ch-fft fine-tuned on BH-TF, with LoRAs

In "more pre-training", we tried adding information to the model by continuing GPT2's pre-training objective of next token prediction on BH-RD. Then we transplanted the new weights into the fine-tuned models, and hoped that they learned new data that way. This would be ideal, as you wouldn't need to create a custom dataset on your model's downstream task to teach it new information.
This got us 4 models:
- bh-gpt2-ch-cht: gpt2-ch-cht pre-trained on BH-RD
- bh-gpt2-ch-fft: gpt2-ch-fft pre-trained on BH-RD
- bhq-gpt2-ch-cht: gpt2-ch-cht pre-trained on BH-RD, with LoRAs
- bhq-gpt2-ch-fft: gpt2-ch-fft pre-trained on BH-RD, with LoRAs

Finally, in "use a KNN", we added the data in BH-TF via the external memory of a KNN. When a KNN model receives a claim, it looks at what it predicts internally, but also the labels of the K nearest claims in its memory. It then combines these two predictions.
This gets us 2 additional models:
- gpt2-ch-cht-knn: gpt2-ch-cht with a KNN model trained with BH-TF
- gpt2-ch-fft-knn: gpt2-ch-fft with a KNN model trained with BH-TF

Once we had all our models, we evaluated their performance on all the datasets and compared which techniques worked and which didn't.

Training Graphs



Results

model name	FEVER train & val accuracy	BH-TF train & val accuracy	SW-TF accuracy
gpt2-ch	0.27 & 0.50	0.52 & 0.50	0.55
gpt2-ch-cht	0.74 & 0.52	0.49 & 0.51	0.47 ↓
gpt2-ch-cht2	0.70 & 0.63 ↑↑	0.55 & 0.56 ↑	0.59 ↑
gpt2-ch-cht2q	0.76 & 0.55	0.50 & 0.53	0.48
bh-gpt2-ch-cht	0.74 & 0.52	0.48 & 0.51	0.45
bhq-gpt2-ch-cht	0.74 & 0.52	0.49 & 0.50	0.46
gpt2-ch-cht-knn	0.63 & 0.61 ↑	0.63 & 0.61 ↑↑	0.54 ↑
gpt2-ch-fft	0.88 & 0.77 ↑↑↑↑↑	0.57 & 0.58 ↑↑	0.55
gpt2-ch-fft2	0.70 & 0.72 ↓	0.96 & 0.71 ↑↑	0.62 ↑
gpt2-ch-fft2q	0.88 & 0.79	0.58 & 0.65 ↑	0.55
bh-gpt2-ch-fft	0.87 & 0.73	0.52 & 0.52	0.50 ↓
bhq-gpt2-ch-fft	0.88 & 0.77	0.57 & 0.59	0.55
gpt2-ch-fft-knn	0.66 & 0.70 ↓	0.66 & 0.70 ↑↑	0.55

Fine-tuning and KNNs were the best techniques to add new information into the model. Pre-training did not help.

Fine-tuning and KNNs, while improving the model's performance on the new data, they hurt the model's performance on the data it had previously memorized. With fine-tuning, we were able to mitigate this by using LoRAs. With KNNs, we did not have a way to mitigate it.

Using LoRAs to mitigate previous knowledge loss only works the first time you add new data - the next time you'll have to update the LoRAs' or the model's weights. (can you use multiple LoRAs?)

Maybe this can be mitigated in KNNs using a dynamic alpha, where it trusts the model more with old data, and the KNN data with new data.

As the models performed poorly on SW-TF - data in GPT2's training - we can guess that the reason pre-training didn't work is because the data learned in pre-training is not retained for the downstream task. Is there a technique that allows the model to learn from Wikipedia articles? Does model size matter? The downstream task?