Fine-Tuned Latent LLMs: A Conceptual Framework

Abstract

Test

Contents

1 Introduction

In recent years, the field of Artificial Intelligence (AI) has witnessed a remarkable evolution in the capabilities of Large Language Models (LLMs). These models, epitomized by systems like GPT (Generative Pre-trained Transformer), have transformed our approach to natural language processing (NLP), offering unprecedented levels of fluency and versatility in language understanding and generation. The ability of LLMs to parse, interpret, and produce human-like text has not only advanced computational linguistics but also opened new frontiers in AI's practical applications, ranging from automated text generation to complex problem-solving.

Despite their impressive abilities, LLMs are not without limitations. A significant challenge lies in the scope and depth of their training. Traditionally, LLMs are trained on vast corpora of text data through 4methods like next-token prediction, which, while effective in building a broad knowledge base, often do not fully equip these models to handle contextually rich or specialized tasks. This gap is particularly evident in scenarios requiring external knowledge integration or step-by-step logical reasoning—areas where even the most advanced LLMs can struggle.

This paper proposes a novel conceptual framework aimed at enhancing the training of LLMs by integrating latent updates—a method that enriches the training process with additional, contextually relevant information generated by the models themselves. These latent updates can take various forms, from generated Python code, which is externally evaluated and then reintegrated into the model's context, to step-by-step reasoning processes that mimic human thought patterns. By embedding these updates into the training cycle, we hypothesize that LLMs can achieve a deeper 4understanding of context and exhibit improved performance in complex tasks.

Our objective is to outline a theoretical model for this enhanced training approach, supported by illustrative pseudo-code and Python examples. We aim to demonstrate how latent updates can be seamlessly integrated into existing LLM architectures and training regimes, potentially leading to more sophisticated and contextually aware AI systems. The remainder of this paper is structured as follows: we first review the standard methodologies in LLM training, followed by a detailed discussion of our proposed method, including its theoretical underpinnings, implementation strategies, and potential impacts on LLM training and applications.

In sum, this paper seeks to contribute to the ongoing discourse in AI and NLP by proposing a method that not only builds upon the existing strengths of LLMs but also addresses some of their key limitations. Through this exploration, we aim to pave the way for more nuanced and capable language models, better equipped to handle the complexities and subtleties of human language and thought.

2 Background

The journey of Natural Language Processing (NLP) and AI has been marked by continuous evolution, with significant milestones shaping the current landscape. Initial efforts in NLP were rule-based systems, focusing on parsing and pattern matching. The emergence of statistical methods brought a shift towards probabilistic models, leading to the first wave of machine learning applications in language tasks.

A pivotal moment in this evolution was the introduction of the transformer architecture, first detailed in the paper "Attention Is All You Need" by Vaswani et al. in 2017. This architecture, with its self-attention mechanisms, marked a departure from previous recurrent neural network (RNN) models. It enabled more efficient processing of sequences and significantly improved performance in various NLP tasks.

The concept of pretraining language models on vast corpora of text data came into prominence with models like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer). These models, particularly GPT and its successors, demonstrated remarkable abilities in generating coherent and contextually rich text, setting new standards for language model capabilities.

LLMs have found diverse applications, ranging from generating human-like text in chatbots to aiding in complex data analysis. Their ability to understand and generate natural language has opened new frontiers in technology, making AI more accessible and versatile in various domains.

This early evolution of NLP and AI, culminating in the development of transformer-based LLMs, sets the foundation for our exploration int enhancing these models further. The remarkable journey from rule-based systems to sophisticated LLMs underscores the field's dynamic nature and the continuous pursuit of more advanced and capable AI systems.

2.1 Standard Training Methodologies for LLMs

Pretraining The pretraining of Large Language Models (LLMs) typically involves training on extensive text corpora. This phase is crucial in developing a broad language understanding and general world knowledge. Models like GPT-3 are pretrained on datasets comprising a wide range of internet text. This diverse exposure enables them to learn language structures, grammar, and a vast array of information, laying the groundwork for their language capabilities.

Fine-Tuning Post pretraining, LLMs undergo fine-tuning, a process where the model is further trained on a more specific dataset tailored to particular tasks or domains. This step adjusts the model to perform specialized functions, from sentiment analysis to legal document review. Fine-tuning adapts the general understanding acquired during pretraining to the nuances and specific requirements of targeted applications.

Challenges in Training LLMs One of the critical challenges in LLM training is striking the right balance between generalization and specialization. While pretraining offers broad knowledge, fine-tuning must ensure that this doesn't come at the expense of losing the model's ability to adapt to specific contexts. The model must retain its general language abilities while honing its skills for specialized tasks.

Despite their extensive training, LLMs face limitations in tasks requiring deep contextual understanding or complex logical reasoning. They may struggle in situations that demand integration of external knowledge or inferences beyond the scope of their training data. This limitation is particularly evident in scenarios that require understanding of novel concepts or complex problem-solving.

The standard training methodologies of pretraining and fine-tuning have been instrumental in the development of LLMs. However, the inherent challenges of these methods, especially in terms of contextual understanding and logical reasoning, highlight the need for innovative approaches in training. This sets the stage for exploring the integration of latent updates as a means to bridge these gaps and enhance the capabilities of LLMs.

3 Theoretical Basis for Latent Data

Traditional LLM training methodologies, primarily based on next-token prediction, have demonstrated significant success in various language tasks. However, these methods often fall short in contexts requiring deep understanding or intricate logical reasoning. This gap highlights the need for a more nuanced approach to enrich the LLMs' training environment.

In human cognition, context plays a crucial role in understanding and responding to language. For LLMs, contextual depth can be enhanced by incorporating additional, relevant information that goes beyond the immediate textual data. This approach aims to mimic the human ability to utilize background knowledge and internal reasoning in language processing.

In the context of AI and LLMs, **latent data** refers to information that is internally generated to aid the model's processing. That is, given a sequence of training tokens, the model enriches it with relevant data. This latent data can take various forms, from the evaluation of generated code to step-by-step reasoning processes. The model's task is to predict the next token based on this enriched context, refining its predictive accuracy and contextual awareness.

To find the optimal parameter θ, we maximize the likelihood of the training data given the model's predictions Y:

$$\theta = \arg\max_{\theta \in \Theta} \mathcal{L}(\theta|Y),$$

where

$$Y = \{\text{predict}(c(X, L), \theta) : X \in D\}$$

is the model's predictions,

$$D = (X_1, X_2, ..., X_m)$$

is the raw training data,

$$X = (x_1, x_2, ..., x_n)$$

is a sequence of tokens in the raw training data,

$$L = \text{gen\_latent}(X, \theta)$$

is the latent data, and

$$c : \text{raw data} \times \text{latent data} \rightarrow \text{context}$$

is a function that combines the latent data with the original data in some way to provide for an enriched context. We are interested in letting c be the defined as

$$c(X, L) = \text{`}< |latent| >\text{`} + L + \text{`} < |latent| >\text{`} + X$$

and we are interested in a number of different latent data generation functions gen_latent:

- L_{code}: generate relevant code based on the training data, guided by the current model parameters. The code is then evaluated externally, and the output is used as the latent data.
- L_{reason}: generate step-by-step reasoning processes based on the training data, guided by the current model parameters. The reasoning process is then evaluated externally, and the output is used as the latent data.
- L_{pred}: generate predictive models or statistical inferences based on the training data, guided by the current model parameters. The predictive models are then evaluated externally, and the output is used as the latent data.

The integration of latent updates represents a significant advancement in the training of LLMs. By enriching the model's context with internally generated, relevant information, this approach aims to bridge the gap in complex contextual understanding and logical reasoning. This theoretical framework sets the foundation for more sophisticated models capable of nuanced interpretation and esponse generation in language tasks.

3.0.0.1 Inferences At inference time, the model is given a sequence of tokens $X = (x_1, x_2, ..., x_n)$ and is asked to continue the sequence. It should be possible to use the same latent data generation function gen_latent as in training, since that is how the model was fine-tuned.

4 Latent Data Generation Functions

When we augment the training data with the latent data, we may use various tools to help the model generate the latent data.

4.1 Code Generation

The first latent data generation function we consider is L_{code}, which generates relevant code based on the training data, guided by the current model parameters. The code is then evaluated externally, and the output is used as the latent data along with either the code or a description of the code (whicih can be generated by the model itself).

> TODO: Mention somewhere that a transformer learns mini-programs that fit into its layers and weights. Also menntion that an AR process increases the efficiency and effectiveness (refer to the paper I have in mind) of learning. Also mention that the latent data generation function can be a neural network itself, and that the latent data can be generated by a neural network that is trained to generate the latent data, or an LLM fine-tuned to. Finally, we can connect these insights to this paper.

> TODO: Does this help with mechanistic interpretation of the model? understanding how it generates python code is still very hard, but at least we get to see the python code it generates, and it can be more easily understood and it is probably easier to therefore align it with human values.

Prompt Engineering

To generate any relevant code, we need to provide the model with a prompt that guides it to generate the code that, upon evaluation, will produce the desired latent data. The pretrained (and fine-tuned) model will already have significant knowledge of Python, so the question is not whether it can generate the code, but rather to use its knowledge of Python to generate appropriate code, or none at all if it is not necessary. This is a non-trivial task that will require some experimentation and problem solving.

Example Prompt You are an autoregressive LLM that has been fine-tuned on Python code. You are presently being trained on a large corpus of text data, using the semi-supervised next-token prediction task. You will be given some raw text data, and you are asked to predict the next token, e.g., a word or a punctuation mark. However, it can sometimes be very difficult to accurately predict the next token, particularly for complex tasks that require deep contextual understanding or algorithmic reasoning, such as mathematical or logical computations. In these cases, it may be easier to generate Python code that can be evaluated externally, and the output of the code can be used as the latent data along with other supplementary information, such as the python code itself and a description of the code.

Let's work on an example to show how this works. Suppose we have the following text:

<text> The names of the students are John, Mary, and Peter. John its 15 years old, Mary is 16 years old, and Peter is 17 years old. Their average age is </text>

You are asked to generate the next token. In order to do so, you may be inclined to generate the Python code:

<python>

```
peter = 17
mary = 16
john = 15
average_age = mean([peter, mary, john])
print(average_age)
```

</python>

An external Python interpreter will evaluate the code, producing the output:

<output>

16

</output>

We prepend this information to the original text, which hopefully provides it with an enriched context in which to more successfully predict the next token in the original text. Here is what the enriched latent context might look like:

<text> <latent> Here is some python code that may help with the next token prediction task. <python>

```
peter = 17
mary = 16
john = 15
average_age = mean([peter, mary, john])
print(average_age)
```

</python> <output>

16

</output>

Recall that the data generating process of reality is impossibly complex, and the output provided by this code is only an approximation of the true output. Keep in mind the larger context of the original raw data, and only use this information to help you predict the next token. </latent> The names of the students are

John, Mary, and Peter. John its 15 years old, Mary is 16 years old, and Peter is 17 years old. Their average age is `</text>`

The model will then likely predict the next token, which is probably `16`. Of course, sometimes, it may be the case that the raw text provides a different continuation, like `given by 16` or even a mathematically incorrect continuation, like `17`. If this is the case, the model will be penalized for its incorrect prediction, and so the task is to let this inform your next-token prediction task, but to also learn to ignore the output or only use it as an additional contextual data point (maybe it is close to 16, for instance).

5 Future Work

We would like to entertain more sophisticated latent data generation functions gen_latent. We are particularly interested in *tree-of-thought* generation, where the latent data is a tree of thoughts that the model can use to guide its predictions. In this way, we can fine-tune the LLM to consider many possible paths of reasoning, and the model can choose the most appropriate path based on the context. This approach is inspired by the human ability to consider multiple paths of reasoning and choose the most appropriate one based on the context. We believe that this approach will lead to more sophisticated models capable of advanced system 2 thinking.