Lu Sun, and many more.

A Notebook on Machine Learning Updates

To my family, friends and communities members who have been dedicating to the presentation of this notebook, and to all students, researchers and faculty members who might find this notebook helpful.

Contents

Foreword

If software or e-books can be made completely open-source, why not a notebook?

This brings me back to the summer of 2009 when I started my third year as a high school student in Harbin No. 3 High School. In around the end of August when the results of Gaokao (National College Entrance Examination of China, annually held in July) are released, people from photocopy shops would start selling notebooks photocopies that they claim to be from the top scorers of the exam. Much curious as I was about what these notebooks look like, never have I expected myself to actually learn anything from them, mainly for the following three reasons.

First of all, some (in fact many) of these notebooks were more difficult to understand than the textbooks. I guess we cannot blame the top scorers for being so smart that they sometimes make things extremely brief or overwhelmingly complicated.

Secondly, why would I want to adapt to notebooks of others when I had my own notebooks which in my opinion should be just as good as theirs.

And lastly, as a student in the top-tier high school myself, I knew that the top scorers of the coming year would probably be a schoolmate or a classmate. Why would I want to pay that much money to a complete stranger in a photocopy shop for my friend's notebook, rather than requesting a copy from him or her directly?

However, things had changed after my becoming an undergraduate student in 2010. There were so many modules and materials to learn in a university, and as an unfortunate result, students were often distracted from digging deeply into a module (For those who were still able to do so, you have my highest respect). The situation became even worse as I started pursuing my Ph.D. in 2014. As I had to focus on specific research areas entirely, I could hardly split much time on other irrelevant but still important and interesting contents.

This motivated me to start reading and taking notebooks for selected books and articles, just to force myself to spent time learning new subjects out of my comfort zone. I used to take hand-written notebooks. My very first notebook was on *Numerical Analysis*, an entrance level module for engineering background graduate students. Till today I still have on my hand dozens of these notebooks. Eventually, one day it suddenly came to me: why not digitalize them, and make them accessible online and open-source, and let everyone read and edit it?

—
As most of the open-source software, this notebook (and it applies to the other notebooks in this series as well) does not come with any "warranty" of any kind, meaning that there is no guarantee for the statement and knowledge in this notebook to be absolutely correct as it is not peer reviewed. **Do NOT cite this notebook in your academic research paper or book!** Of course, if you find anything helpful with your research, please trace back to the origin of the citation and double confirm it yourself, then on top of that determine whether or not to use it in your research.

This notebook is suitable as:

- a quick reference guide;
- a brief introduction for beginners of the module;
- a "cheat sheet" for students to prepare for the exam (Don't bring it to the exam unless it is allowed by your lecturer!) or for lecturers to prepare the teaching materials.

This notebook is NOT suitable as:

- a direct research reference;
- a replacement to the textbook;

because as explained the notebook is NOT peer reviewed and it is meant to be simple and easy to read. It is not necessary brief, but all the tedious explanation and derivation, if any, shall be "fold into appendix" and a reader can easily skip those things without any interruption to the reading experience.
—

Although this notebook is open-source, the reference materials of this notebook, including textbooks, journal papers, conference proceedings, etc., may not be open-source. Very likely many of these reference materials are licensed or copyrighted. Please legitimately access these materials and properly use them.

Some of the figures in this notebook is drawn using Excalidraw, a very interesting tool for machine to emulate hand-writing. The Excalidraw project can be found in GitHub, *excalidraw/excalidraw*.

Preface

Artificial intelligence (AI) and specifically machine learning were originally a research topic under the scope of control systems, and it is primarily used for system identification ("machine learning was used to be called system identification", as some professors say). Its artificial neuron network (ANN) structure is a promising approach for building highly nonlinear self-tuning functions.

Due to the limitations of computational power and data storage in the past years, training large-size ANN with massive data points was hardly possible. With the advancement of computer and material science in the beginning of the 21st centry, in particular the development of GPU and TPU, nowadays we can manage networks with dozens of layers, each containing hundreds of neurons. This is referred as the deep learning (DL) network structure. With DL as the building block, effective convolutional neural networks (CNN) and recurrent neural networks (RNN) have been proposed which are effective at finding trends in spacial and sequential data. The proposition of transformers, yet another attention-based deep learning structure, has taken natural language processing to the next level.

AI has grown so significantly in the past decades that it is now considered a separate area apart from classic control systems, and it is drawing more and more attentions. To put it in perspective, while a well-known textbook on control systems may have 10,000+ citations records after years since its publication, a famous conference paper on modern AI can easily have 50,000+ citations after a few months.

This notebook does not focus on the introduction of AI mechanisms, as most of which are already covered in control systems related notebooks. The purpose of this notebook is mainly to keep up-to-date with the latest technology in machine learning, starting from transformer and its applications in chatbot, the most famous of which being ChatGPT developed by OpenAI. The notebook cares not only the techniques, but also the business models used in these technologies and companies.

Some part of this notebook is written by ChatGPT-4.

List of Figures

List of Tables

Part I

Technologies

1

Transformer

CONTENTS

This chapter introduces the development of "transformer" in its main branch. Transformer is an AI framework initially proposed in 2017 in a paper "Attention is All You Need", and it has magically revolutionized how we train AI for natural language processing and interpreting.

1.1 It Seems That Attention is All You Need

It started in 2017 when "transformer" is proposed in a paper "Attention is all you need", by Vaswani, et al., a bunch of guys mainly from Google.

"Attention" was not a new concept previous to this paper. It was proposed a few years earlier than the transformer, and had been integrated into several CNN and RNN models to boost their performances to capture long-range dependencies. In these researches, attention mechanism is more often a supporting role than the main character. It is quite surprising indeed, that when getting rid of CNN and RNN and purely relying on attention mechanism, everything actually gets better: it is faster to train the network, and its performance gets improved.

Does this mean the RNN, which we had been relying on so much, is in fact the drawback to natural language processing? The following is the answer of ChatGPT-4 of this question.

Q: By getting rid of RNN and using attention mechanism alone, natural language processing performance gets better. Is this correct? Does this mean that RNN is a drawback to natural language processing?

A: Yes, by getting rid of RNNs and using the attention mechanism alone, the Transformer model demonstrated improved performance in natural language processing tasks. However, it's important to note that RNNs are not inherently a drawback to NLP; they were successfully used for many tasks before the advent of Transformers.

RNNs, particularly LSTM (Long Short-Term Memory) and GRU (Gated Recurrent Unit) variants, were designed to handle sequences and capture dependencies in the data. They have been effective in many NLP tasks, such as language modeling, machine translation, and sentiment analysis. However, RNNs do have certain limitations, such as difficulty in capturing long-range dependencies and sequential processing, which can lead to slower training and inference times.

The attention mechanism, as employed in the Transformer model, addresses some of these limitations. It enables parallel processing of input sequences, leading to faster training and inference, and effectively captures long-range dependencies without the vanishing gradient problem that can affect RNNs.

While Transformers have shown superior performance in many NLP tasks, RNNs still have their place in certain scenarios, especially when working with resource-constrained environments or smaller datasets. They can be more lightweight and computationally efficient compared to large-scale Transformer models.

In summary, RNNs are not a drawback to NLP per se, but their limitations have been mitigated by the introduction of the attention mechanism in Transformer models, which has led to improved performance in a wide range of NLP tasks.

(April 14, 2023, ChatGPT-4)

In the remaining part of the chapter, a quick review of CNN and RNN is given and their limitations introduced. Attention mechanism is illustrated. The original transformer model is explained. Finally, research trends of transformer as of today are briefly summarized.

1.2 CNN, RNN, and Their Limitations

This section does not focus on the detailed introduction of CNN and RNN mechanisms. Instead, only their main features are reviewed, and their limitations discussed. This section does not reflect on the state-of-art of researches in CNN and RNN.

The conventional feedforward network (network without cycles) with dense ANN is widely used. It can be trained systematically using back-propagation methods such as stochastic gradient descent. However, dense ANN is not very good at interpreting correlations among the inputs. At least, it is not good at doing it efficiently.

For example, consider the input to be pixels of an image of size $3 \times 128 \times 128$, where 3 corresponds to the RGB of the image, and 128×128 its pixel size. It is obvious to a human that the pixel at $(1, 1)$ should definitely has a closer relationship with the one at $(1, 2)$ (they are probably forming an object in the image together) than $(128, 128)$. The same applies to sequential inputs. Consider a signal sampled continuously. The first and second samples are very likely to be more strongly correlated than the first and last samples.

In a conventional dense ANN, the correlation cannot be captured efficiently. In contrast, a dense ANN would treat all the inputs "equally" in a symmetric manner. To enforce the ANN to "memorize" the correlation, it would require a lot more layers and nodes (which is often considered low-efficient), and requires more data points during the training. Other problems of conventional dense ANN include, for example, the lack of ability in handling data with arbitrary length.

CNN and RNN try to tackle the above problems by implementing a "pre-processing" stage, where the correlation of the spacial and sequential data is first abstracted using some mechanism, and the correlation information is sent as (additional) inputs to the followed dense ANN.

The details of CNN and RNN mechanisms are not discussed in this notebook. Brief reviews are given as follows.

1.2.1 Brief Review of CNN

CNN is a type of ANN structure designed to handle grid-like data. It is effective when dealing when data spatially correlated, thus becomes very popular in computer vision. It defines "kernel" that aggregates nearby pixels information before sending it to a dense network.

A CNN kernel, also known as a filter, is a small matrix of weights that slides across the input image or feature map to perform a mathematical operation called convolution. An demonstration of CNN kernel is given in Fig. 1.1. Multiple kernels can be defined on the same layer to handle the same feature map, each kernel associated with an output channel. In practice, each kernel or channel is designed to detect a specific features in the feature map. For example, there might be a kernel detecting edges, while a second kernel detect color codes.

Notice that CNN differs quite largely from transformer in the problems they are expected to address. CNN is more for spatial data processing such

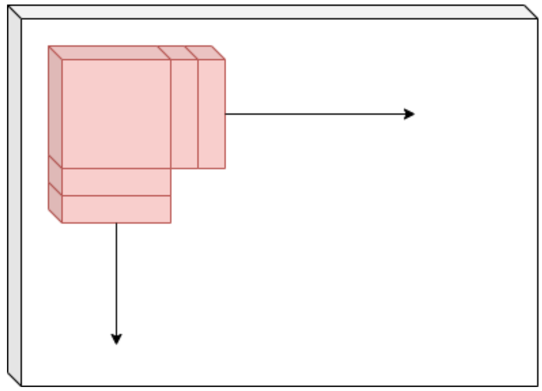

FIGURE 1.1
A demonstration of CNN kernel. The input is given by the white box (3D), and the kernel by the red box.

as image processing, while transformer targets more on sequential data processing such as natural language processing and machine translation.

1.2.2 Brief Review of RNN

RNN is a connectionist model with the ability to selectively pass information across sequence steps [2]. It is good at handling sequence of data such as voice message, text contents, or a flow of images (videos). It is worth mentioning that the "sequence" does not necessarily mean a time sequence. Nevertheless, without losing generality, we will consider time sequence in the review for simplicity and convenience.

Denote inputs $x(1), x(2), ..., x(k), ...$ where $x(k)$ is a vector sampled at time instant k. The length of the sequence may be finite or infinite. In the case of finite sequence, its maximum sample index is denoted by T. For example, in the context of natural language processing, each input might be a word in a dictionary. For example, $x(1)$ = "Pandas", $x(2)$ = "are", $x(3)$ = "so", $x(4)$ = "cute", $x(5)$ = "!". The corresponding target output sequence is given by $y(1), y(2), ..., y(k), ...$, respectively.

RNN differs from the conventional dense ANN by introducing "recurrent edges", which allows the output of hidden layers at $k-1$ be used as additional inputs to the system at k. This means, at any time k, the input of the system includes both $x(k)$ and also selected $h(k-1)$, where $h(\cdot)$ is the outputs of hidden layers. We can think of the "weights" of a trained RNN the "long-term memory" that does not change with specific sequence of inputs, while the information passing through recurrent edges the "short-term memory" that links previous inputs with future inputs.

A demonstration is given in Fig. 1.2. Notice that each hidden layer is a

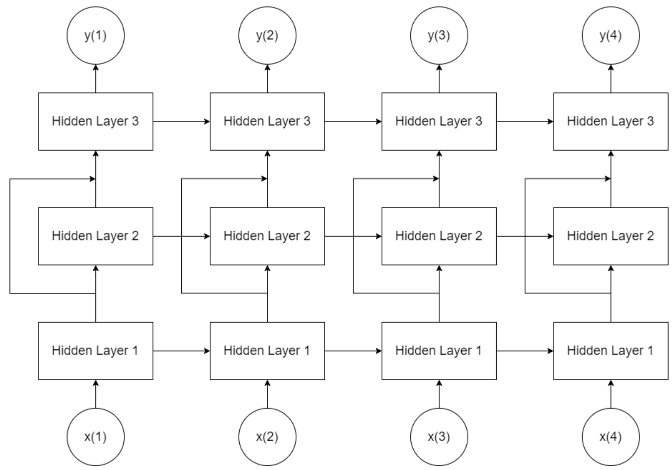

FIGURE 1.2
A example of RNN.

multi-input-multi-output subsystem containing multiple nodes. Different from a conventional dense ANN, each hidden layer takes additional inputs from its corresponding hidden layer in the previous instant. It is also common to see "bypass" (this is widely used in different ANN structures, not unique to RNN) for better performance of the system.

RNN has some limitations. One of the major problem is that it is difficult to train an RNN even for the basic standard feedforward networks. The optimization of RNN is NP-complete. It is especially difficult for RNN to learn long-range dependencies due to the vanishing and exploding gradients problem that could occur when backpropagating errors across many timestamps (long sequence) [2]. This is one of the main challenges why RNN has difficulties building on long-range dependencies. The vanishing and exploding gradient problems are caused by the structure of the system as well as the backpropagation-based training methods.

Different approaches have been proposed to prevent vanishing and exploding gradient problems. Famous ones among these approaches include strategical weight initialization, long short term memory (LSTM), gated recurrent units (GRUs), skip connections, and more. Many of these approaches try to reduce the effect of vanishing and exploding gradient problems by carefully design the ANN structures. For example, both LSTM and GRUs introduce "memory cells" with built-in "gates" that balance and control the flow of information from previous cells versus current inputs. The cells are used to replace the traditional perceptron nodes. These approaches have made the training of RNN a feasible problem. LSTM and GRUs are almost certainly used in modern RNNs.

Bidirectional RNN (BRNN) is proposed at about the same time with

LSTM. It allows information to travel not only from previous hidden layers to future layers, but also from future hidden layers to previous layers. LSTM and BRNN can be used together to boost the RNN performance. Notice that BRNN cannot run continuously as it requires fixed endpoints in both the future and the past. It is useful for prediction over a sequence of fixed length, such as part-of-speech tagging in natural language processing.

Another problem that people have found during the training of RNN is local optima. However, recent studies have shown that local optima is not as serious issue as we might thought when the network is large, since many critical points are actually saddle points rather than local minima.

Successful implementations of the above RNN structures include natural language translation such as [4] where a encoder-decoder structure is used, each is a LSTM. Another example is image captioning, where the AI tries to explain what is in an image using texts. A solution to this is to use CNN to encode the image, and use LSTM to decode to generate texts. Following similar ideas is hand-writing reorganization.

1.3 Attention Mechanism

“nobreak

1.4 Transformer

While CNN is mainly used for spatial data processing such as computer vision, both RNN and transformer are mainly used for sequential data processing. It is worth mention in the very beginning that transformer does not guarantee superior performance in all scenarios comparing with traditional RNN-based natural language processing models such as LSTM.

For one thing, transformer consumes larger computational capabilities. Transformer processes data in batch, while RNN does them in sequence, meaning that RNN can response faster in some real-time applications. It is difficult for the transformer to process super-long text due to the computational burden (quadratic computational complexity with respect to sequence length), while RNN can process arbitrarily long text, although the later suffers from capturing long-range dependencies.

With the above been said, the transformer does demonstrated superior performance than RNN in many tasks. The mechanism and the reasons why it performs better in these occasions are introduced as follows.

1.5 Research Trends

Part II

Products

Many data companies, including Microsoft, Google, Meta (formerly known as Facebook) and Amazon, provide their own data models and AI engines of different kinds. Famous and AI engines and models include TensorFlow by Google, PyTorch by Meta, and ChatGPT by OpenAI, which is a company supported by Microsoft.

These data companies usually open (part of) the source code of these models and engines in the community and encourage individual developers to learn and use them. At the same time, they provide enterprise level services such as model fine-tuning to make profit from the tools.

Some state-of-art models, engines, as well as the business models of these companies are reviewed in this part of the notebook.

2

OpenAI

CONTENTS

“nobreak

2.1 General Introduction

“nobreak

2.1.1 Background

“nobreak

2.1.2 Business Model

“nobreak

2.2 ChatGPT

“nobreak

2.3 OpenAI API

“nobreak

2.4 DELL-E-2

3

Google AI

CONTENTS

“nobreak

3.1 General Introduction

“nobreak

3.1.1 Background

“nobreak

3.1.2 Business Model

“nobreak

3.2 TensorFlow

“nobreak

3.2.1 Summary of Capability

“nobreak

3.2.2 Examples

“nobreak

3.3 Bard

4

Meta AI

CONTENTS

Meta, formerly known as Facebook, provides variety of solutions for AI, including computer vision, stream data processing, natural language processing, and many more.

4.1 General Introduction

“nobreak

4.1.1 Background

“nobreak

4.1.2 Business Model

“nobreak

4.2 PyTorch

PyTorch, similar with TensorFlow, is a Python-based generic AI engine developed by Meta. It is easy to deploy and has been widely used in variety of applications including computer vision and sequential signal processing. PyTorch is freely accessible can be installed on customer machines.

4.2.1 Summary of Capability

“nobreak

4.2.2 Examples

“nobreak

4.2.3 PyTorch Lightning

AI engines have tried hard to make themselves easy to use. Ultimately, the AI solution architect should only concern about the network architecture and not the programming. PyTorch Lightning, a PyTorch add-on provided by Lightning.AI, is one of the many attempts to make that happen. PyTorch Lightning comes in the form of an additional Python package.

Notice that PyTorch packages and libraries are still required, as PyTorch Lightning is an add-on, but not a replacement, to PyTorch. PyTorch changes only the way of programming, but not the capability or performance of the models.

Examples are given in *pytorchlightning.ai* where PyTorch programs with and without PyTorch Lightning are compared. The examples show that PyTorch Lightning is helpful with simplifying the pipeline deployment and tiding the programming logic.

4.3 LLaMA

LLaMA is the large language model published by Meta AI. The associated paper that introduced LLaMA for the first time was published in February 2023 on ArXiv as [5]. The initial LLaMA is trained on publicly available datasets exclusively.

Different levels of complexity of the models are provided. Like many other LLMs, Meta AI provides charged service for fine-tuning the model. There is also a re-write of LLaMA, namely Lit-LLaMA, that is completely open source under Apache 2.0 license. Both LLaMA and Lit-LLAMA can be fine-tuned via tools such as LoRA and Stanford Alpaca.

Comparing with other models such as ChatGPT-3, LLaMA is claimed to use smaller models (with less number of parameters) to achieve the same performance.

4.3.1 LLaMA: Open and Efficient Foundation Language Models

LLaMA is another implication of (modified) transformer proposed in [6]. A GitHub repository has been created and made public about LLaMA. The repository is given here: *github.com/facebookresearch/llama*.

Budget-Oriented LLM Design

There is a trad-off between using large model versus small model, given a fixed amount of budget to create and run the model. A large model often implies better performance potentially. But this is true only if it is trained on large amount of data. Otherwise, over-fitting may happen, which increases the variance of the system. A large model plus a large amount of data is sometimes economically inefficient.

Given a fixed budget, comparing using large model with small amount of data which gives only mediocre performance, it might be wise to use a relatively smaller model, but and spend the remaining budget training it with a large amount of data. It is possible to match the performance of the large but under-trained model using the small but well-trained model. The later approach would often mean longer training time, but less resources at the same time and lower execution cost in its daily run post training.

LLaMA implements the later approach. It assumes different levels of fixed budgets, and discusses what might be the optimal choice of LLM under each budget.

Data Sources

Only publicly available data is used in the training. The data sources for LLaMA is summarized Table 4.1. The total amount of training data is about 4.75TB, or 1.4T tokens after tokenization. To put it into perspective, all the work that Shakespeare has done, if converted to plain text, is about 5MB (1/950000 of 4.75TB).

Model

Different sizes of models are tested. The size of the model, i.e., number of parameters, are 6.7, 13.0, 32.5, and 65.2 billion respectively, which is summarized in Table 4.2. Notice that the number of parameters listed above is significantly smaller than many of the available models.

TABLE 4.1
Data sources for LLaMA.

Data Source	Percentage	Description
Common Crawl	67%	Common Crawl English data is used. Common Crawl builds and maintains an open repository of web crawl data that can be accessed and analyzed by anyone. The data can be used for research, analysis, and education.
C4 Dataset	15%	C4 is a colossal, cleaned version of Common Crawl's web crawl corpus. It was based on Common Crawl dataset.
Github	4.5%	Github repositories under Apache, BSD and MIT licenses are used.
Wikipedia	4.5%	Wikipedia data with different Latin and Cyrillic scripts are used.
Public domain books	4.5%	Free ebooks from Gutenberg project, etc., are used.
ArXiv	2.5%	Latex files from ArXiv are used.
Stack Exchange	2%	Stack Exchange is a website of high-quality questions and answers of variety of domains.

TABLE 4.2
LLaMA models by size.

Model Name	Number of Parameters	Data Size for Training
LLaMA 7B	6.7 billion	1T token
LLaMA 13B	13.0 billion	1T token
LLaMA 33B	32.5 billion	1.4T token
LLaMA 65B	65.2 billion	1.4T token

Notice that 1.4T token of training data is approximately 4.75TB of plain text data.

The model architecture is based on the original transformer proposed in [6], with following modifications:

- Add pre-normalization to the input of each transformer sub-layer, instead of normalizing the output of the layers.
- Use SwiGLU activation function instead of ReLU.
- Use rotary positional embedding instead of absolute positional embedding. Rotary Position Embedding, or RoPE, is a type of position embedding which encodes absolute positional information with rotation matrix and naturally incorporates explicit relative position dependency in self-attention formulation. See [3] for details.

AdamW optimizer with dynamic learning rate is used.

Mini-batch training of 4M tokens are used for all models. Models with size 6.7B 13.0B are trained with 1.0T tokens, and the rest are trained with all 1.4T tokens, as shown in Table 4.2. Variety of methods are implemented to speed up the training.

Performance Evaluation

The benchmarks to evaluate and compare the performance of LLaMA with other models are mainly from the following categories:

- Common sense reasoning. E.g., "If someone is holding an umbrella, what might be the reason?"
- Closed-book question answering. E.g., "What is the capital city of France?"
- Reading comprehension. E.g., give a paragraph that introduces generic search, then ask "what is the major advantage of generic search?"
- Mathematical reasoning. E.g., "what is the sum of the first 5 prime numbers?"
- Code generation. E.g., "Generate a code in Python that calculates the first 100 values in Fibonacci series."
- Multitask language understanding. E.g., "Translate English into Franch, then give a summary."
- Bias, toxicity and misinformation.

LLaMA has shown improved performance compared with the model with similar size.

Training Effort

Take the largest LLaMA-65B as an example. An "A100-80GB" GPU (cost approx 15k USD) is used and trained using 1022k GPU-hours. The total power consumption is 449MWh, which is worth approximately 90k USD household price.

4.3.2 LLaMA Services

LLaMA is not yet a fully commercialized AI model, and its code as given in the LLaMA Github repository is under GPL license. Research team behind LLaMA suggests that the primary intended uses of LLaMA should be research on LLMs. The model (at least version 1 of the model) is a base or foundational model, and should not be used on downstream applications without any tuning and mitigation of risks.

The instruction to download and use LLaMA can be found in its Github repository. In short, the user needs ask approval to use the trained LLaMA model. The user needs to setup prerequisite Python environment in order to run the model. With the approval, the user can download the trained LLaMA model and the tokenizer, and run the model on its own machine. An example in Python is provided, which helps the user to verify the downloaded items, test the environment, and get a quick start.

Variety of tools and projects have been developed, either as a demonstration of LLaMA use case, or as a add-on to make the use of LLaMA more convenient. Examples include LoRA, a tool by Microsoft that can be used to fine-tuning models, and Lit-LLaMA, an open-source recreation of LLaMA, and a few more. Details are introduced in later sections.

4.3.3 LLaMA-Relevant Use Cases, Tools, and Projects

LoRA

LoRA (Low-Rank Adaptation of Large Language Models) is an open-source tool developed by Microsoft that enables transfer learning /knowledge distillation/model fine-tuning of LLMs. This is quite a important tool, as training LLMs from scratch can be very time and energy consuming.

LoRA is first proposed in [1]. A Github repository *github.com/microsoft/LoRA* contains the source code of LoRA implementation in Python package `loralib`, which currently support PyTorch. Indeed, LoRA can be used to fine-tune ChatGPT (as under Microsoft), LLaMA and Lit-LLaMA.

The latest version of LoRA is supported by *github.com/huggingface/peft*, which is a collection of state-of-the-art parameter-efficient fine-tuning methods.

Lit-LLaMA

Lit-LLaMA is an open-source recreation of LLaMA by Lightning.AI. Think of Lightning.AI as something similar with GNU project (Linux project). GNU wants to create a open-source version of UNIX that can be freely distributed. Companies like Red Hat can make a profit from it by providing enterprise level support to the subscribers who uses RHEL. Lightning.AI is essentially trying to do the same: it creates user-friendly interfaces for existing AI models such as PyTorch, and sometimes recreates open-source version of existing (commercialized) models, and make them open-source. It encourages users to use their interfaces and models. It makes profits by providing add-on services such as training models using their servers, etc.

Lit-LLaMA is a recreation of LLaMA model based on nanoGPT, another open-source transformer model. Different with LLaMA which is under GPL license, nanoGPT is under MIT license and Lit-LLaMA Apache license, both of which more friendly for private projects. Lit-LLaMA Github repository is given here *github.com/Lightning-AI/lit-llama*.

Unlike LLaMA where approval is required to download the trained model, under the scope of Lit-LLaMA, it is fine to download the trained 7B model freely. The customer can run the trained model on its server. A decently powerful GPU is required. Such a power GPU often causes a few thousands (e.g., NVIDIA GeForce RTX 3090 24GB, which can be used to run and fine-tune the model, approx 3k) to tens of thousands (e.g., NVIDIA A100 80GB, which can be used to train LLaMA and Lit-LLaMA from scratch, approx 15k) USD.

Both LLaMA and Lit-LLaMA provides interfaces and example scripts for the users to use the model on their own server. In addition, they also provide interface for LoRA, which can be used to fine-tune the model. Python script examples are provided to use LoRA on Lit-LLaMA in the repository.

Stanford Alpaca

It is quite unfortunate that, in the area of AI, academic researchers and universities have fallen behind. The most powerful models are usually trained using very expensive resources including high-cost GPU clusters, months or years of human labors, and tens or hundreds of terabytes of data. It is often done by data driven companies who would not make them completely open-source to the public even for research purposes. LLaMA is an outlier in the good sense, and it gives Stanford a chance to make a break through.

Stanford Alpaca 7B is a fine-tuned model from LLaMA 7B model. The cost of the fine-tuning is impressively small (hundreds of USD), and the performance of the system after fine-tuning is comparable with ChatGPT-3.5 (text-davinci-003) which has a significantly larger model of 335B than 7B.

It is common that a fine-tuned model gets its performance boosted. However, it is quite surprising to see a huge boost of performance happening on LLaMA 7B, as this model is famous for being small and already being trained by a huge amount of data. How much potential does LLaMA 7B have that can be further improved? Not to mention, everything is done using only a few hundred USD, given that a GPU alone that can train the model would usually cause thousands of dollars.

It is worth mentioning that text-davinci-003 outputs are used to fine-tune LLaMA 7B to create Stanford Alpaca 7B. This may imply:

- Training Stanford Alpaca 7B is essentially fine-tuning a less-power model (LLaMA 7B) using the outputs from a more powerful model (text-davinci-003), instead of using plain text. This may explain partially why it is so inexpensive, as data collection and preprocessing is usually one of the most expensive parts in AI training and it is now taken care of by text-davinci-003.

- It is suspicious whether the performance of LLaMA 7B can surpass text-davinci-003 in general, if it is trained by text-davinci-003 in the first place. This is because the bias of information can be enlarged during training.

- The above suggests that very likely other techniques than training with text-davinci-003 been used to further improve its performance.

An introduction of Stanford Alpaca is given here *crfm.stanford.edu*. An introduction to using the model is given in its Github repository *github.com/tatsu-lab/stanford_alpaca*. As of April 2023, Stanford Alpaca has released the fine-tuning receipt, and it claims that the trained model will be released in the future.

Bibliography

[1] 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, 2021.

[2] Zachary C. Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning, 2015.

[3] Jianlin Su, Yu Lu, Shengfeng Pan, Ahmed Murtadha, Bo Wen, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding, 2022.

[4] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.

[5] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[6] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2017.