Lu Sun, and many more.

A Notebook on Artificial Intelligence

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 **vii**

Preface **ix**

List of Figures **xi**

List of Tables **xiii**

I Introduction **1**

II Searching **3**

III Reasoning and Planning **5**

IV Neural Networks and Machine Learning **7**

1 Regression **9**

2 Perceptron **11**

3 Multi-layer Perceptron **13**

V Computer Vision **15**

4 Convolutional Neural Network **17**
- 4.1 Brief Review of CNN 17

VI Sequential Data Processing **19**

5 Recurrent Neural Network **21**
- 5.1 Brief Review of RNN 21

6 Transformer **25**
- 6.1 Attention is All You Need 25
- 6.2 Limitations of Conventional Methods 26
- 6.3 Attention Mechanism 27
- 6.4 Transformer 27
- 6.5 Research Trends 28

7 Large Language Model 29
- 7.1 Introduction 30
  - 7.1.1 Language Models 30
  - 7.1.2 Scaling Law 31
  - 7.1.3 Emergent Abilities 32
  - 7.1.4 Milestone Techniques 33
- 7.2 Existing Resources and Solutions 34
  - 7.2.1 Data Corpus 34
  - 7.2.2 Libraries and Packages 35
  - 7.2.3 OpenAI Family 35
  - 7.2.4 LLaMA Family 37
- 7.3 LLM Training Pipeline 43
  - 7.3.1 Architecture Design 43
  - 7.3.2 Training Set Preparation 43
  - 7.3.3 Pre-training 43
  - 7.3.4 Fine-Tuning 43
  - 7.3.5 Model Evaluation 44
- 7.4 Multimodal LLM 44
- 7.5 Examples 44
  - 7.5.1 LLM with Fine-Tuning 44
  - 7.5.2 Semantic Search 44
  - 7.5.3 Database Powered LLM 44
  - 7.5.4 Multimodal LLM for Signal Processing 44
  - 7.5.5 Multimodal LLM for Image Processing 44

Bibliography 45

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) was included as part of the control system notebook, as in early ages it was mostly used as a system identification tool to support a controller.

With the advent of graphical processing units (GPU) in the 1990th and the Industry 4.0 initiatives in 2000th, deep learning network with massive training data has become possible, which significantly boost the performance of the artificial neural network (ANN) based AI systems. Nowadays, AI has been growing rapidly with more and more successful demonstrations of useful cases such as computer vision and natural language processing.

Seeing that trend, AI relevant contents have been split from the control system notebook, and put here.

List of Figures

4.1 A demonstration of CNN kernel. The input is given by the white box (3D), and the kernel by the red box. 18

5.1 A example of RNN. 22

7.1 A cat with superpower. This picture is generated by DALL·E 3. 37
7.2 OpenAI DALL·E 3 draws cats with superpower. 38
7.3 LLaMA family tree. 39
7.4 Alpaca fine-tuning pipeline. 40

List of Tables

7.1 Existing LLM models. 34
7.2 LLaMA models. 39

Part I

Introduction

Part II

Searching

Part III

Reasoning and Planning

Part IV

Neural Networks and Machine Learning

1

Regression

CONTENTS

2

Perceptron

CONTENTS

3

Multi-layer Perceptron

CONTENTS

Part V

Computer Vision

4

Convolutional Neural Network

CONTENTS

4.1 Brief Review of CNN 17

“nobreak

4.1 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. 4.1. 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 as image processing, while transformer targets more on sequential data processing such as natural language processing and machine translation.

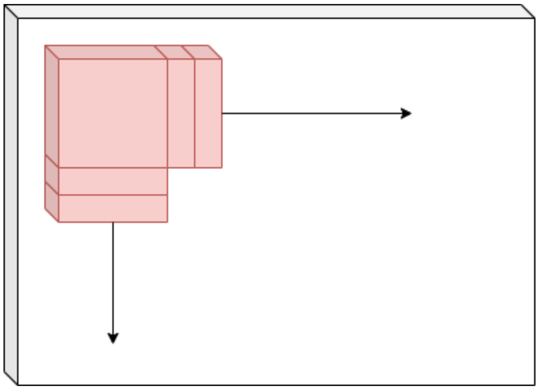


FIGURE 4.1
A demonstration of CNN kernel. The input is given by the white box (3D), and the kernel by the red box.

Part VI

Sequential Data Processing

5

Recurrent Neural Network

CONTENTS

5.1 Brief Review of RNN 21

“nobreak

5.1 Brief Review of RNN

RNN is a connectionist model with the ability to selectively pass information across sequence steps [4]. 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. 5.1. Notice that each hidden layer is a 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

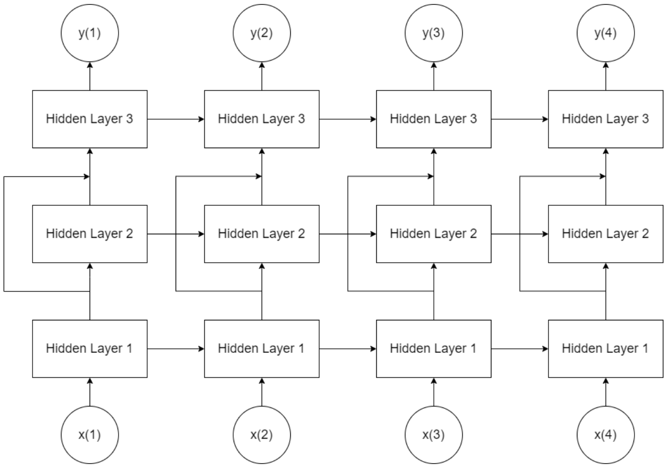


FIGURE 5.1
A example of RNN.

"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) [4]. 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 [7] 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.

6

Transformer

CONTENTS

6.1 Attention is All You Need 25
6.2 Limitations of Conventional Methods 26
6.3 Attention Mechanism 27
6.4 Transformer 27
6.5 Research Trends 28

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.

6.1 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.

6.2 Limitations of Conventional Methods

CNN and RNN have been introduced previously. 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 $3\times128\times128$, 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.

6.3 Attention Mechanism

“nobreak

6.4 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.

6.5 Research Trends

7

Large Language Model

CONTENTS

7.1 Introduction 29
  7.1.1 Language Models 30
  7.1.2 Scaling Law 31
  7.1.3 Emergent Abilities 32
  7.1.4 Milestone Techniques 33
7.2 Existing Resources and Solutions 34
  7.2.1 Data Corpus 34
  7.2.2 Libraries and Packages 34
  7.2.3 OpenAI Family 35
  7.2.4 LLaMA Family 37
7.3 LLM Training Pipeline 43
  7.3.1 Architecture Design 43
  7.3.2 Training Set Preparation 43
  7.3.3 Pre-training 43
  7.3.4 Fine-Tuning 43
  7.3.5 Model Evaluation 43
7.4 Multimodal LLM 44
7.5 Examples 44
  7.5.1 LLM with Fine-Tuning 44
  7.5.2 Semantic Search 44
  7.5.3 Database Powered LLM 44
  7.5.4 Multimodal LLM for Signal Processing 44
  7.5.5 Multimodal LLM for Image Processing 44

This chapter serves as an overview of large language model (LLM), a machine learning model that can handle natural language, read and learn from text documents, and communicate with humans. The focus of this chapter is mostly on high-level motivations, features, basic mechanisms and general training pipeline of LLM.

Detailed introductions to the LLM development platforms (Google Colab, etc.), programming languages and packages (Python, PyTorch package, Transformer package, etc.), and existing solutions (GPT, LLaMA, etc.), will be given in later chapters separately.

The majority of this chapter comes from [10].

7.1 Introduction

There have been variety of ways to model natural language, some of which have been introduced in previous chapters. For example, consider the following sentence:

"I am thirsty. Please give me a bottle of ______."

It is quite natural that a human would likely to put "water" or "tea" in the blank. This is obvious because human has a dictionary of words that he can choose in his mind, and he has been reinforced of learning "a bottle of water" expression in many occasions. In addition, it makes sense to a human that when someone is thirsty, he would look for water.

The challenge using AI to realize the above is to build the dictionary in the machine and quantitatively analyze what word or phrase would make the most sense to be filled into the blank. For that, many models have been developed.

7.1.1 Language Models

The most popular language models have been evolving over time in the past decades since 1990th.

Statistical Language Models

In the early days when AI and ANN were not popular, *statistical language models (SLM)* have been the most popular tool to model natural language. SLM assumes that a sentence is a Markov process, and the last word depends on the context created by the most recent n words. SLM with a fixed context length of n is also called a n-gram language model.

Conventionally, the paring information from n words to the next word is obtained from data corpus and stored in a table-like structure. When running the model, it looks up the table for the most probable next word based on the earlier n words. Smoothing technologies are used to handle zeros, i.e., when the record is not found in the table.

An obvious issue of SLM is that the computation and storage of the model increase exponentially with the size of n. This limits the context information that the model can use for prediction, hence setting a low performance ceiling. Not to mention that even with a large data corpus, zeros can still happen and the model performance is always an issue in such occasions.

Neural Language Models

With the introduction of ANN, in particular RNN, *Neural Language Models (NLM)* become popular. RNN-based NLM builds the word prediction function conditioned on the aggregated context features abstracted and passed recurrently from current and previous input sequence.

RNN is not a perfect one-stop solution either. The training of RNN can be difficult due to vanishing and exploding gradient problem. This limits the depth that RNN can go. When handling long sequence of words, the performance of RNN drops significantly because it is weak at building long-term dependencies.

Pre-trained Language Models and Large Language Models

As introduced in details in earlier chapters, attention mechanism has been proposed to tackle the long-term dependency problem of RNN. Transformer architecture, which relies purely on encoder, decoder and attention mechanism without using RNN is then proposed. It has been verified that transformer architecture is good at abstracting information from sequential data, in particular, natural language. With transformer, it becomes possible to build very deep neural networks and have it trained efficiently with big size data corpus.

Language models based on different transformer-based architectures are often called pre-trained language models (PLM) and large language models (LLM). The main differences between PLM and LLM are the size of the model. The scaling of the model from hundreds of millions of parameters (PLM) to tens or hundreds of billions of parameters (LLM) introduces emergent abilities such as in-context learning to the model, significantly enhancing its capability and intelligence. As of this writing, it is not very clear how the these abilities suddenly emerge with the size of the model.

7.1.2 Scaling Law

The performance of an LLM, usually referring to its capability in accurately and correctly complete a task, is affect by many factors such as the model architecture, model size, training data set size and quality, etc. Though it is clear that with the scaling up of the system the performance is usually improved, there is no analytical expression that gives full insights about how these factors affect the performance quantitatively.

Many scaling laws have been proposed trying to quantitatively describe the LLM performance as a function of its model size and other factors. Notice that these laws are obtained from empirical experiments and they may work only within a given range of model size.

Just as an example, OpenAI proposed KM scaling law in 2020 that describes LLM cross entropy loss as a function of model size, training data set size and training computation as follows.

$$
\begin{aligned}
L(N) &= \left(\frac{N_c}{N}\right)^{\alpha_N} \\
L(D) &= \left(\frac{D_c}{D}\right)^{\alpha_D} \\
L(C) &= \left(\frac{C_c}{C}\right)^{\alpha_C}
\end{aligned}
$$

where N, D and C denote the model size, dataset size and training computation, respectively. The rests are constants whose value can be obtained via calibration.

This scaling law works for models with $22M$ to $23B$ parameters. It is assumed that the analysis of a factor can be done independently without other parameters being a bottleneck.

7.1.3 Emergent Abilities

When the size of the model becomes large, usually to the order of at least a few billions parameters, they suddenly gain emergent abilities. Details are discussed as follows.

In-context Learning (ICL)

ICL allows the behavior of the model be manipulated via not training or fine-tuning of the parameters, but via instructions and demonstrations given as part of the input.

ICL plays an important part in LLM implementation, as it is the basis of prompt engineering. When the LLM is large, it is possible to use prompt engineering instead of fine-tuning for it to complete a task following user defined instructions. This reduces the training cost and makes the implementation more flexible.

Instruction Following

Supervised learning is commonly used in fine-tuning an LLM. Examples include providing “ideal responses” for different types of questions, so that LLM would know how to respond to these questions.

When comes to LLM, it is actually possible to fine-tune the model for a task without presenting it examples. Instead, just give it step-by-step instructions. LLM is able to perform well with these tasks described only by instructions. This is known as instruction tuning.

It is worth mentioning that instruction following is also possible in ICL. Give the LLM instructions in prompt engineering without examples, and the LLM is likely to be able to finish the tasks.

Step-By-Step Reasoning

When a model is asked to complete a complicated task that involves multiple steps, it may fail to accomplish the task. With a well fine-tuned LLM, the model might be able to break the task into multiple sub-tasks via chain-of-thought (CoT) prompting strategy, and solve them step-by-step till the final result is obtained.

It has been observed that LLM with $100B$ parameters or more, and have been trained on code is likely to have good step-by-step reasoning ability.

7.1.4 Milestone Techniques

This section looks back into the progression tree of LLM, and list down milestone techniques that make LLM what it is today. It is the breakthrough in these areas that revolutionizes LLM development.

Big Data

The performance of LLM relies on both the modal size and the training data size. It is the advent in internet, Web 3.0, Industry 4.0, IoT and cloud computing/storage that makes collection and aggregation of big data possible.

Almost every large-size enterprise, both IT companies or conventional industrial companies, has its internal database. The database can be used to train domain LLM. Nowadays, there are many open data sources of community LLMs. Such examples include BookCorpus, CommonCrawl, Reddit posts (with high upvotes), Wikipedia, and many more. These open-source datasets makes training LLM for community projects possible.

Large Model and Efficient Training

The invention of CPU-based neural networks and transformer architecture making creating and training large scale LLM possible. Both closed and open-source LLMs have been proposed, including GPT series by OpenAI and LLaMA family by Meta AI and the community.

Manly libraries have been released to the public to help building, training and fine-tuning LLMs, such as `transformers`, a Python library for building transformer models. Many such libraries are tying up with PyTorch and TensorFlow to provide LLM related functions.

More about these models and libraries are introduced in later sections.

Fine-Tuning

Technologies such as LoRA has made fine-tuning easier than before.

Prompt Engineering

There have been a lot of practices on how to make LLM flexible and more efficient in solving particular tasks via prompt engineering.

LLM on Edge Devices

Many efforts have been put into edge-device based LLMs. The target is to develop LLM that consumes less memory, storage and computation while not sacrificing a lot of performance.

API and Interface

Multi-modal LLM has enabled different types of inputs to the LLM, not limited to natural language but also sequential signals and even pictures.

TABLE 7.1
Existing LLM models.

Availability	Model Name	Size	Training	Release Time
open	CodeGen	16	$577B$	2022-Mar
	LLaMA	65	$1.4T$	2023-Feb
	CodeGen2	16	$400B$	2023-May
	LLaMA2	70	$2T$	2023-Jul
closed	GPT-3	175	$300B$	2020-May
	Codex*	12	$100B$	2021-Jul
	GPT-4	—	—	20223-Mar

Model size is given in number of parameters in billion. Training data size is given in number of tokens.
* Codex is trained on top of GPT-3.

Many tools and software have developed APIs for LLM. These tools enhance computation and online information retrieval capabilities of LLM.

7.2 Existing Resources and Solutions

Many data corpus, both open source and closed source, are available on the market. Many libraries such as PyTorch compatible Python packages have been developed to automate the training procedures and to convenient the users. Many LLM, both open-source and closed-source, have been proposed. A summary is given in Table 7.1. Notice that the table only covers a small portion of existing models in the market.

The existing resources and solutions are briefly introduced in this section.

7.2.1 Data Corpus

Enterprise and individual users may have their own closed source data corpus for training and fine-tuning. In this section, only open source data corpora are discussed.

7.2.2 Libraries and Packages

“nobreak

7.2.3 OpenAI Family

OpenAI started investigating language models before the proposition of transformer. In its early days, RNN was explored as the most promising model for natural language. In 2017 when the transformer model was proposed, OpenAI quickly adapted their language model to this new architecture, and as a result generative pre-training (GPT) series has been proposed.

GPT-1

GPT-1, OpenAI's first transformer based PLM was proposed in 2018. GPT-1 has $117M$ parameters in the model and it adopts a decoder-only architecture, which is different from the original transformer proposal which has a encoder-decoder architecture. GPT-1 was trained via a two-stage procedure, the first stage unsupervised pre-training and the second stage supervised fine-tuning. This two-stage training pipeline, or something of the similar kind, has been adopted by many LLMs coming after.

GPT-2

GPT-2 is an improvement of GPT-1. It uses much larger number of parameters of $1.5B$ in the model, and it was trained on a much larger dataset WebText. With larger model and training data size, GPT-2 is targeted to be a multi-task solver. The model can be formulated by the following probabilistic form

$$\text{Pre-trained LLM} \quad \equiv \quad P\left(\text{output}|\text{input}, \text{task}\right)$$

In the above formulation, each NLP task can be considered as the word prediction problem based on a subset of the word next, and can be trained during the unsupervised learning stage. Unsupervised pre-training has since then become the most important stage for the LLM to gain knowledge for general tasks.

GPT-3, GPT-3.5 and Chat-GPT

It is clear now that GPT-2 has $1.5B$ parameters which is too few for an LLM to gain emergent abilities. It is GPT-3 with $175B$ parameters trained on $300B$ tokens that made a capability leap and bring LLM to everyone's attention.

It is GPT-3 that for the first time introduces emergent abilities such as ICL. GPT-3 not only accomplishes commonly seen tasks to test LLM capabilities with flying color, but also demonstrates features not shown by other models before, such as reasoning and domain adaption.

OpenAI has developed many task-oriented models that use GPT-3 as the base model. For example, for coding, Codex was introduced. Codex is basically GPT-3 fine-tuned using code database such as GitHub. Comparing with GPT-3, Codex is able to reason and solve complex mathematical problems, and realize them in codes. RL had already been used to fine-tune and improve performance for GPT-2. The same has been applied on GPT-3. Furthermore, reinforcement learning with human feedback (RLHF) is introduced for GPT-3 that allows the model to continue learning from human demonstrations.

With the above enhancements, i.e. code-based fine-tuning, RL and RLHF, GPT-3.5 has been developed. GPT-3.5 is an enhanced version of GPT-3 and it is obtained from GPT-3 via transfer learning. GPT-3.5 is also scaled up in size, with a parameter number of $335B$.

Chat-GPT was developed on top of GPT-3.5 (and later another version was released developed on top of with GPT-4). It is GPT-3.5 fine-tuned specially optimized for dialogue. It demonstrates how LLM can be used in everyone's daily life as an assistant, consultant and a copilot. Chat-GPT demonstrated good performance in communicating of humans than other GPT models. This is because Chat-GPT is trained on human-generated conversations of both human and AI roles. Later on, plug-ins and APIs are enabled on Chat-GPT, to allow it to communicate with other software.

GPT-4

GPT-4 is a multimodal LLM, meaning that it can not only take in natural languages, but also other signals such as images. Comparing with GPT-3.5, GPT-4 demonstrates further enhanced capabilities. It is claimed that GPT-4 is safer to use and is less likely to generate toxic and misleading information.

Just as a quick demonstration of its multimodal feature, pass the following Fig. 7.1 to GPT-4 with instruction "What is in this picture?". GPT-4 is able to response the following:

"The image showcases a majestic cat standing in a fantastical landscape. The cat emits a luminous, ethereal glow, and its fur and tail appear to be infused with swirling, fiery or cosmic energies in brilliant hues of red and gold. These radiant swirls surround the cat, adding to its supernatural aura. Behind the cat, there's a dramatic night sky with clouds illuminated by a hidden light source, possibly lightning or a celestial event. In the foreground, the terrain appears to be a combination of grassy knolls and fiery, ember-lit ground, further emphasizing the magical and otherworldly nature of the scene. The entire setting evokes a sense of wonder and fantasy."

Other OpenAI Models

DALL·E is the model OpenAI uses to generate images. As of this writing, its latest version, DALL·E 3, has been integrated with the latest ChatGPT. Figure 7.2 gives an example where DALL·E 3 is used to generate a image of a superpower cat.

OpenAI Codex, as introduced earlier, is the GPT model optimized for code generation. As of this writing, Codex has been deprecated because its capability has been integrated into ChatGPT. It is possible to ask ChatGPT to generate a piece of code following user's instruction and description.

FIGURE 7.1
A cat with superpower. This picture is generated by DALL·E 3.

OpenAI Codex is the engine that powers GitHub copilot. In this sense, OpenAI Codex remains a live in a different format: not as a standalone software or model, but more as an API.

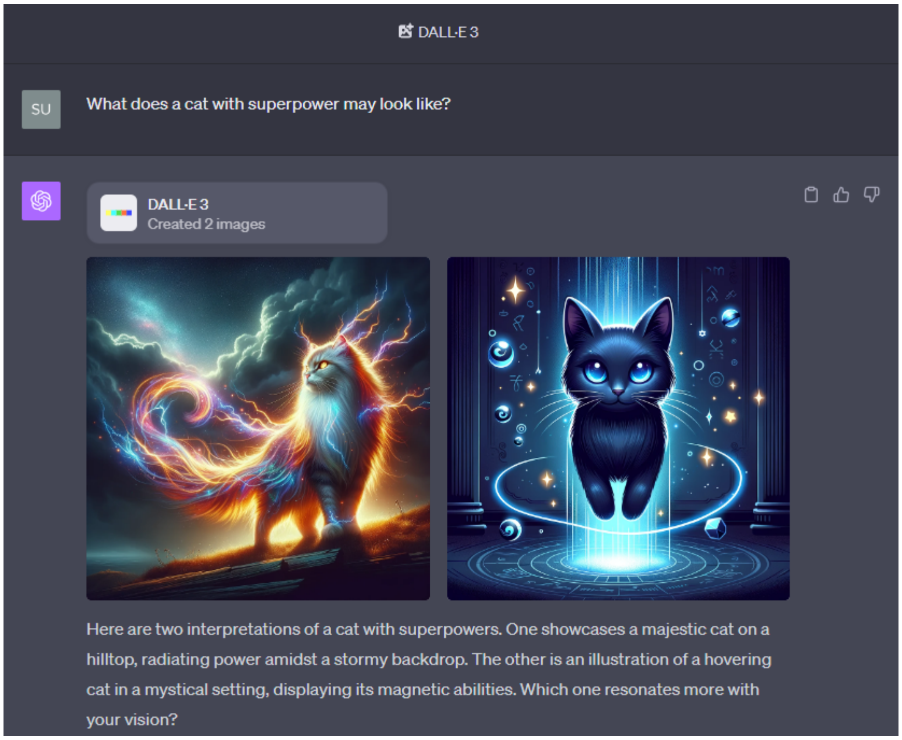
7.2.4 LLaMA Family

Large Language Model Meta AI (LLaMA) is the LLM model developed by Meta AI. Different from most of the AI models (GPT-3 and onward) developed by OpenAI, LLaMA is open-source hence has a wide availability. Many efforts in the community have made modifications and improvements to LLaMA, making a big family of models with different characteristics.

As of this writing, a family tree of LLaMA is shown in Figure 7.3. The picture is from [10]. The source of the picture is given in the GitHub repository of the paper. Only a small portion of models in the family tree is briefly introduced here.

LLaMA

LLaMA, comparing with GPT-3 which was used as a benchmark, is smaller in model size (maximum $65B$ parameters VS $175B$ parameters in GPT-3) but larger and better in training data size and quality (maximum $1.4T$ tokens VS $300B$ tokens in GPT-3). As a result, LLaMA is able to achieve generally better performance than GPT-3 with less implementation cost due to the small size. LLaMA demonstrates that training data is equality important as model size.



What does a cat with superpower may look like?

Here are two interpretations of a cat with superpowers. One showcases a majestic cat on a hilltop, radiating power amidst a stormy backdrop. The other is an illustration of a hovering cat in a mystical setting, displaying its magnetic abilities. Which one resonates more with your vision?

FIGURE 7.2
OpenAI DALL·E 3 draws cats with superpower.

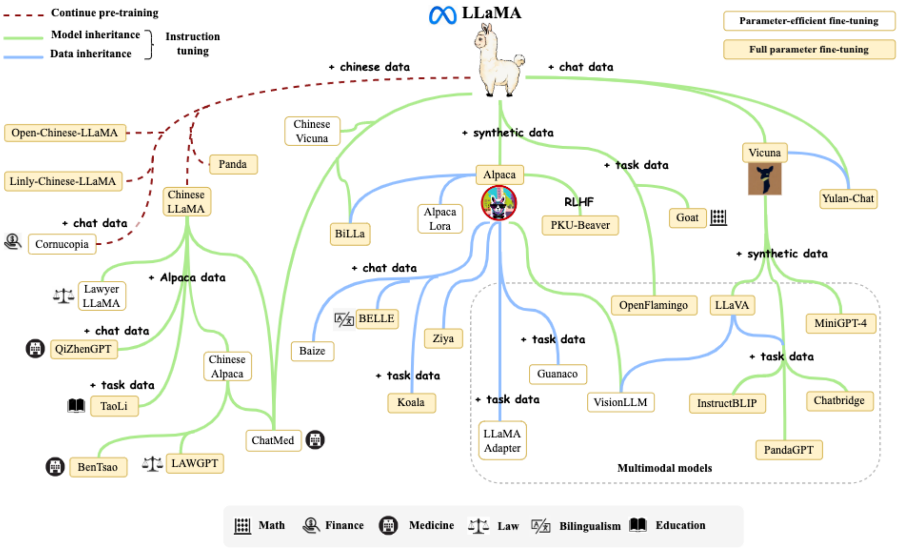


FIGURE 7.3

LLaMA family tree.

It is possible to reach the same level of performance with a small ($< 100B$ parameters) but well-trained model.

LLaMA's first release includes 4 models of different model and training sizes. Details are summarized in Table 7.2. The training dataset is purely open-source, including Common Crawl, C4 Dataset, GitHub, Wikipedia, public domain books, arXiv and Stack Exchange.

Technical wise, LLaMA has some innovations on top of the original transformer proposition [9] n the normalization method, activation function, and embedding methods. More details will be introduced in later sections.

Many open-source tools and packages have been developed to fine-tune LLaMA and its variations. More about fine-tuning, such as LoRA[3] and

TABLE 7.2

LLaMA models.

Name	Model Size	Training Dataset Size
LLaMA 7B	$6.7B$	$1T$
LLaMA 13B	$13.0B$	$1T$
LLaMA 33B	$32.5B$	$1.4T$
LLaMA 65B	$65.2B$	$1.4T$

Model size is given in number of parameters in billion. Training data size is given in number of tokens.

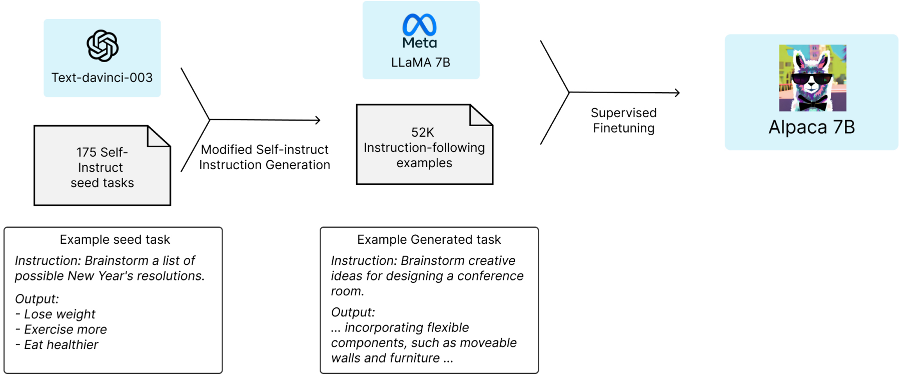

Text-davinci-003
175 Self-Instruct seed tasks
Modified Self-instruct Instruction Generation
Meta
LLaMA 7B
52K Instruction-following examples
Supervised Finetuning
Alpaca 7B
Example seed task
Instruction: Brainstorm a list of possible New Year's resolutions.
Output:
- Lose weight
- Exercise more
- Eat healthier
Example Generated task
Instruction: Brainstorm creative ideas for designing a conference room.
Output:
... incorporating flexible components, such as moveable walls and furniture ...

FIGURE 7.4
Alpaca fine-tuning pipeline.

QLoRA[2], are introduced in more details in later sections. Notice that fine-tuning or re-training of a model often cost a lot of computational power and vector memory.

Stanford Alpaca

Stanford Alpaca 7B is Standford's practice in fine-tuning LLaMA 7B. The running cost of this fine-tuning is impressively small (hundreds of USD), and the resulted model has a performance comparable with GPT-3.5 which has $335B$ parameters. It is impressive to see that a small $7B$ model can compete with a large $335B$ model by careful training and fine-tuning. This might be partially because Alpaca 7B is fine-tuned from LLaMA 7B using knowledge distillation from GPT-3.5 (also known as text-davinci-003).

The details of training Stanford Alpaca can be found in [8]. A brief highlight is given below. The fine-tuning pipeline of Alpaca is shown in Fig. 7.4. It is a 2-stage process as follows.

1. Generating instruction-following examples using GPT-3.5 (text-davinci-003).
2. Fine-tune LLaMA 7B using the examples.

Examples of the generated instruction and response pairs are given below. The full list is available from the GitHub repository.

```
{
      "instruction": "What are the three primary colors?",
      "input": "",
      "output": "The three primary colors are red, blue, and yellow."
},
{
      "instruction": "Give three tips for staying healthy.",
      "input": "",
      "output": "1.Eat a balanced diet and make sure to include plenty
           of fruits and vegetables. \n2. Exercise regularly to keep
          your body active and strong. \n3. Get enough sleep and
          maintain a consistent sleep schedule."
},
{
      "instruction": "Extract the facts from the paragraph.",
      "input": "Online education continues to become more popular for
          schools and students alike. Its advantages are generally
          lower costs, less commitment and the ability to study at a
          time, place and pace that suits the student.",
      "output": "- Online education is becoming increasingly popular.\
          n- It has several advantages such as lower costs, less
          commitment and the ability to study at one\u2019s own time
          and pace."
},
{
      "instruction": "Arrange the given numbers in ascending order.",
      "input": "2, 4, 0, 8, 3",
      "output": "0, 2, 3, 4, 8"
}
```

The fine-tuning code of Alpaca is from Hugging Face. Python packages such as `numpy`, `torch`, `transformers`, etc., are required. Check `requirements.txt` and `train.py` in the repository for more details. Call `train.py` using something like the following

```
torchrun --nproc_per_node=4 --master_port=<your_random_port> train.py \
        --model_name_or_path <
            your_path_to_hf_converted_llama_ckpt_and_tokenizer> \
        --data_path ./alpaca_data.json \
        --bf16 True \
        --output_dir <your_output_dir> \
        --num_train_epochs 3 \
        --per_device_train_batch_size 4 \
        --per_device_eval_batch_size 4 \
        --gradient_accumulation_steps 8 \
        --evaluation_strategy "no" \
        --save_strategy "steps" \
        --save_steps 2000 \
        --save_total_limit 1 \
        --learning_rate 2e-5 \
        --weight_decay 0. \
        --warmup_ratio 0.03 \
        --lr_scheduler_type "cosine" \
        --logging_steps 1 \
        --fsdp "full_shard auto_wrap" \
        --fsdp_transformer_layer_cls_to_wrap ’LlamaDecoderLayer’ \
        --tf32 True
```

where `./alpaca_data.json` is the JSON file that contains all the instruction and response pairs.

Notice that later on, Stanford Alpaca 13B is also developed based on LLaMA 13B.

Vicuna

Vicuna 13B is an open-source chatbot fine-tuned on top of LLaMA using $70K$ user-shared conversations collected from ShareGPT (a platform where users share their ChatGPT conversation histories). Its performance is comparable with GPT-3.5 and Bard, and it is slightly better than LLaMA 13B and Alpaca 13B in general.

See [1] for more details.

LLaVA and MiniGPT-4

Large Language and Vision Assistant (LLaVA) is a multimodal LLM that combines a vision encoder and Vicuna for general purpose visual and language understanding. More details are given in [6] and [5]. Similar with GPT-4, its multimodal encoders allows it take both natural language and image as inputs. An example is given below.

Pass the same Fig. 7.1 to LLaVA with the same instruction "What is in this picture?". LLaVA is able to give the following response:

"The image features a cat standing on a hill, with a fiery background and a cloudy sky. The cat appears to be looking at the viewer, and there is a sense of motion in the scene."

Similar with LLaVA, MiniGPT-4 is a also multimodal LLM that enhances vision interpretation. More details are given in [11]. It uses different techniques and realizations from LLaVA. Pass the same Fig. 7.1 to MiniGPT-4 with the same instruction "What is in this picture?". MiniGPT-4 is able to give the following response:

"The image shows a white cat with flames coming out of its fur, standing on a hill at sunset. The cat's eyes are closed, and it appears to be in a meditative state. The background is a dark sky with clouds and a full moon. The overall mood of the image is peaceful and mystical."

Notice that both LLaVA and MiniGPT-4 model can be downloaded from their GitHub and HuggingFace repositories. As of this writing, live demos are also available for these models.

Multimodal LLM and "Any-to-Any" LLM are not the main forecast of this chapter. More details will be introduced in later chapters.

LLaMA 2

LLaMA 2 is the next generation open source LLM following the original

LLaMA. The model is free for research and commercial use. LLaMA 2 70B has $70B$ parameters and it is pre-trained with $2T$ tokens of training data.

7.3 LLM Training Pipeline

“nobreak

7.3.1 Architecture Design

Encoder and Decoder

Normalization Method

Activation Function

Position Embedding Method

Attention Mechanism

7.3.2 Training Set Preparation

Commonly used data corpora include the following.

7.3.3 Pre-training

“nobreak

7.3.4 Fine-Tuning

“nobreak

7.3.5 Model Evaluation

“nobreak

7.4 Multimodal LLM

“nobreak

7.5 Examples

“nobreak

7.5.1 LLM with Fine-Tuning

“nobreak

7.5.2 Semantic Search

“nobreak

7.5.3 Database Powered LLM

“nobreak

7.5.4 Multimodal LLM for Signal Processing

“nobreak

7.5.5 Multimodal LLM for Image Processing

Bibliography

[1] Wei-Lin Chiang, Zhuohan Li, Zi Lin, Ying Sheng, Zhanghao Wu, Hao Zhang, Lianmin Zheng, Siyuan Zhuang, Yonghao Zhuang, Joseph E. Gonzalez, Ion Stoica, and Eric P. Xing. Vicuna: An open-source chatbot impressing gpt-4 with 90%* chatgpt quality, March 2023.

[2] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *arXiv preprint arXiv:2305.14314*, 2023.

[3] 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.

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

[5] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2023.

[6] Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning. In *NeurIPS*, 2023.

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

[8] Rohan Taori, Ishaan Gulrajani, Tianyi Zhang, Yann Dubois, Xuechen Li, Carlos Guestrin, Percy Liang, and Tatsunori B. Hashimoto. Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`, 2023.

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

[10] Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 2023.

[11] Deyao Zhu, Jun Chen, Xiaoqian Shen, Xiang Li, and Mohamed Elhoseiny. Minigpt-4: Enhancing vision-language understanding with advanced large language models. *arXiv preprint arXiv:2304.10592*, 2023.