自主學習報告 – LLM(Large Language Model)初探

參與者:吳庭佑	地點:北大高中	時間:2025/3/15~2025/4/1
操作方式:實驗操作		
簡述:透過對 LLM 的研究，我重新審視了自己整理資料、設計實驗、找到解決方法的過程。 無論是接下來大學的專題、論文的撰寫，這次的過程都將成為我的經驗和養分，而不用在 大學端重新學習。		

目錄

一、 研究動機

生成式 AI 正逐漸滲透進每一個人的生活中，對業界、學界皆帶來一定的衝擊。所以為了和 AI 可以做到更好的協作，我想研究生成式 AI 的架構和實作生成式 AI，並最後希望能轉化成自己可以使用的 AI 工具。

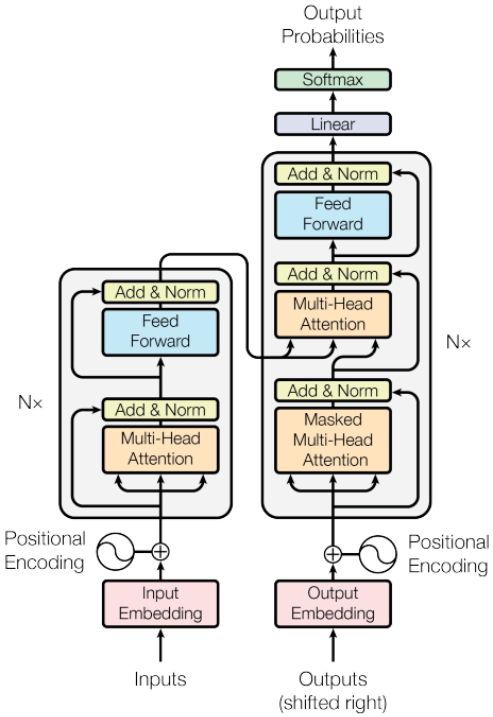
二、 背景知識

Transformer

由<<Attention Is All You Need>>[1]這篇論文所提出，其使用 Encoder-decoder 結構[2]:encoder 將輸入(input)的序列$(x_1, \dots, x_n)$標註為連續的序列$\boldsymbol{z} = z_1, \dots, z_n$，而 decoder 根據序列$\boldsymbol{z}$生成輸出(output)序列$(y_1, \dots, y_n)$，而模型的這些步驟皆擁有自回歸(auto-regressive)的特性，也就是將前個時間步t的輸出y_t作為下個時間步$t+1$的輸入。

[1] https://doi.org/10.48550/arXiv.1706.03762

[2] https://doi.org/10.48550/arXiv.1406.1078



The Transformer – 模型結構

Encoder

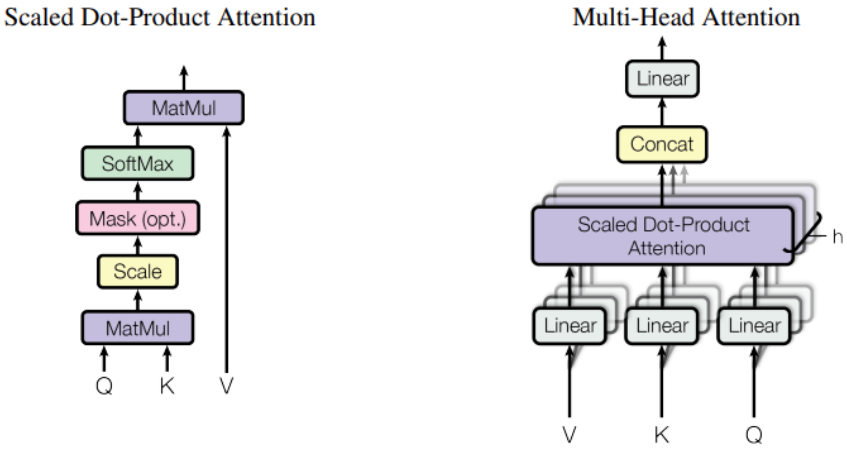
在<<Attention Is All You Need>>這篇論文中，Transformer 的 Encoder 使用$N = 6$層相同的編碼器(identical layers)組成，每層都有兩個子層(sub-layers)組成。第一子層是一個 Multi-Head Self-Attention，第二子層是一個簡單的 Position-wise Fully Connected Feed-forward Network，子層和子層間使用 Residual Connection 後接著 Layer Normalization，意味著每個子層的輸出為$LayerNorm\big(x + Sublayer(x)\big)$。為了使 Residual Connection 可行，所有層的輸出維度必須相同，因此固定$d_{model} = 512$。

Decoder

在<<Attention Is All You Need>>這篇論文中，Transformer 的 Decoder 同樣使用$N = 6$層相同的編碼器組成，每層除了原本的兩個子層，更加入了第三個子層 Multi-performs Attention，加入這層的原因是為了防止 Decoder 的子層提前注意到了還未解讀的內容，子層和子層間同樣使用 Residual Connection 後接著 Layer Normalization。

Attention

這是一種函數，它會根據一個 query vector 和一組 key-value pairs，產生一個 output vector。Output vector 是通過 values 的加權總合得到，而每個值的加權由 compatibility function 得到。



Scaled Dot-Product Attention

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)\text{V}$$

d_k是 Key 向量的維度，$\sqrt{d_k}$是以 query 和所有 keys 做 dot-product 得到的結果，矩陣 Q 是矩陣包裝的注意力函數，K 為 keys vector，V 為 values vector。

Multi-Head Attention

$$MultiHead(Q, K, V) = Concat(head_1, \dots, head_h)W^O$$

$$where\ head_i = Attention\left(QW_i^Q, KW_i^K, VW_i^V\right)$$

h是計算注意力的次數，意味著它有h個 head，使這個模型可以關注到不同表現的資訊。每個 head 都有自己的線性投影矩陣，將原始d_{model}維的 query、key 和 value 映射到更小的維度(d_k, d_k, d_v)，則以上過程可以用以下矩陣表示:

$$W_i^Q \in R^{d_{model} \times d_k}, W_i^K \in R^{d_{model} \times d_k}, W_i^V \in R^{d_{model} \times d_v}, and\ W^O \in R^{hd_v \times d_{model}}$$

Position-wise Feed-Forward Networks with ReLU

$$FFN(x) = \max(0, \text{x}\text{W}_1 + \text{b}_1)\text{W}_2 + \text{b}_2$$

FNN 的輸入和輸出維度$d_{model} = 512$，而內部隱藏層的維度$d_{ff} = 2048$。

Embeddings and Softmax

Embedding 是將 Input tokens 和 Output tokens 轉換成向量的技術，而 Softmax 則是將 Output vector 轉換成機率的函數。Transformer 採用了一樣的參數矩陣在 Embedding Matrix 和

Pre-Softmax Linear Transformation 中，並將 Embedding 的權重乘上$\sqrt{d_{model}}$。

Positional Encoding

在<<Attention Is All You Need>>這篇論文中，因為 Transformer 模型沒有 RNN 或 CNN 結構，所以需要 PE 來讓模型理解序列的順序。於是 Transformer 模型在 Input embeddings 的最後加入了 Positional encodings，來讓模型知道位置的關係。在 Transformer 模型中使用 sine 和 cosine 函數達成這個目的:

$$PE_{pos,2i} = sin\left(pos/10000^{2i/d_{model}}\right)$$

$$PE_{pos,2i+1} = cos\left(pos/10000^{2i/d_{model}}\right)$$

pos 是位置，i 是維度。

BERT

引用資料:

https://haren.medium.com/paper-notes-bert-bert-%E6%9E%B6%E6%A7%8B%E7%90%86%E8%A7%A3-31c014d7dd63

https://arxiv.org/pdf/1810.04805

全名:Bidirectional Encoder Representations from Transformers

使用 Masked LM(MLM)和 Next Sentence Prediction(NSP)捕捉 token-level 和 sentence-level 的特徵。訓練方式為透過遮住某個詞語來讓模型預測被遮住的詞語。

GPT

引用資料:

https://reurl.cc/mxGN0A

全名:Generative Pre-trained Transformer

GPT 僅採用 Transformer decoder 的部分，透過前面的文字序列來預測下一個單詞，或透過周圍的語境來推斷缺失單詞的意思。

PaLM

引用資料:

https://arxiv.org/pdf/2204.02311

全名:Pathways Language Model

PaLM 採用完整的 Transformer 架構，並使用 Pathways 架構訓練，使其可以同時學習多種不同的任務。

LLaMA

引用資料:

https://ithelp.ithome.com.tw/m/articles/10366983

全名:Large Language Model Meta AI

LLaMA 僅採用 Transformer decoder 的部分，相比於 Transformer 的 LayerNorm，它使用

RMSNorm，且使用更加平滑的 SwiGLU 函數作為激活函數。LLaMA 是開源的，每個人都可以訓練自己的模型。

AutoMathText

引用資料: https://arxiv.org/pdf/2402.07625

此資料集是一個經過精心整理的資料集，包含約 200GB 的數學文字。這個資料集來自不同平台，包含各種網站、arXiv 和 GitHub(OpenWebMath、RedPajama、Algebraic Stack)等。這個豐富的資料庫是由最先進的開放原始碼語言模型 Qwen-72B 所自主選擇 (標籤)。資料集中的每篇內容都會在 [0, 1] 的範圍內分配一個分數 lm_q1q2_score，以反映其在數學智慧方面的相關性、品質與教育價值。

三、 實驗操作

為了方便訓練模型，本次研究選擇在 Colab 平台上訓練模型，Colab 平台提供了免費的 GPU 資源，亦提供安裝好的虛擬環境，意味著使用者不需要自行設定虛擬環境便可使用。

而訓練的模型和數據則從 Hugging Face 上取得，Hugging Face 整合了大量的模型的 API 和訓練數據，以方便使用者操作。

本次實驗使用 GPT2 模型進行操作。首先將要用的模組引入:

```
import math
import numpy as np
import matplotlib.pyplot as plt
import time
import tensorflow as tf
import tensorflow_text as text
```

取得數據集:

```
from datasets import load_dataset

# 載入 train split，隨機選取 100 筆資料
subset_size = 100
dataset = load_dataset("math-ai/AutoMathText", split="train")
subset_dataset = dataset.shuffle(seed=42).select(range(subset_size))

# 以 80%/20% 切割成訓練集與測試集
split_dataset = subset_dataset.train_test_split(test_size=0.2, seed=42)
train_dataset = split_dataset["train"]
test_dataset = split_dataset["test"]
print("訓練集筆數:", len(train_dataset))
print("測試集筆數:", len(test_dataset))
```

引入模型:

```
from transformers import AutoModelForCausalLM, TrainingArguments, Trainer

# 確保使用適當的模型
model_name = "gpt2-large"
model = AutoModelForCausalLM.from_pretrained(model_name)
```

設定 tokenizer:

```python
def tokenize_function(examples):
    return tokenizer(examples["text"], truncation=True, padding="max_length", max_length=128)

tokenized_train = train_dataset.map(tokenize_function, batched=True, remove_columns=["url", "date", "meta"])
tokenized_test = test_dataset.map(tokenize_function, batched=True, remove_columns=["url", "date", "meta"])
```

訓練參數設置:

```python
training_args = TrainingArguments(
    output_dir="./gpt2-automathtext",    # 模型與 checkpoint 儲存目錄
    overwrite_output_dir=True,
    num_train_epochs=3,
    per_device_train_batch_size=16,
    evaluation_strategy="steps",   # 依照步數進行評估
    save_steps=500,                # 每 500 步保存一次 checkpoint
    logging_steps=100,
)

# 確保數據集有 labels
def add_labels(example):
    example["labels"] = example["input_ids"][:]
    return example

tokenized_train = tokenized_train.map(add_labels)
tokenized_test = tokenized_test.map(add_labels)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_train,
    eval_dataset=tokenized_test,
)

# 開始訓練
trainer.train()
```

評估和儲存模型:

```
eval_metrics = trainer.evaluate()
print("評估結果：", eval_metrics)
if "eval_loss" in eval_metrics:
    perplexity = math.exp(eval_metrics["eval_loss"])
    print("Perplexity:", perplexity)

trainer.save_model()
```

用模型生成數學題目的回覆:

```
from transformers import pipeline

# 建立 text-generation pipeline
generator = pipeline("text-generation", model=model, tokenizer=tokenizer)

# 定義 10 個數學問題（難度從簡單到困難）
questions = [
    "What is 7 + 5?",
    "What is 9 x 6?",
    "Solve for x: 2x + 3 = 11.",
    "Simplify the fraction 24/36.",
    "Find the derivative of f(x) = x^2 + 3x - 5.",
    "Solve the quadratic equation: x^2 - 5x + 6 = 0.",
    "Evaluate the integral ∫0^1 x^3 dx.",
    "Find the eigenvalues of the matrix [[2,1],[1,2]].",
    "Solve the differential equation dy/dx = y * tan(x) with y(0)=1.",
    "Discuss the implications of the Riemann Hypothesis on the distribution of prime numbers."
]

print("\n模型對以下數學問題的生成回應：\n")
for idx, q in enumerate(questions):
    print(f"問題 {idx+1}: {q}")
    # 生成答案，設定 max_length
    result = generator(q, max_length=100, num_return_sequences=1)
    for output in result:
        print("生成回應：", output["generated_text"])
    print("="*60)
```

四、 實驗結果

本次研究總共設定了 10 個數學問題來測試訓練過後的 GPT2 模型，難度從簡易到難都有，但除了第一個問題有實際回答出來以外，其他的回覆皆沒有得到想要的結果:

模型對以下數學問題的生成回應：

問題 1: What is 7 + 5?

生成回應： What is 7 + 5?

7 + 5 = 14

$$14 + 5 = 20$$

==========

問題 2: What is 9 x 6?

生成回應： What is 9 x 6?

The first question you should ask is, "what is 9 x 6?"

8 x 9 = 60

10 x 9 = 60

11 x 9 = 60

So, " 9 x 6 " is the ratio of a number to the number of digits.

9 x 6 = 60 x 10 = 60

9 × 6 = 60

==========

問題 3: Solve for x: 2x + 3 = 11.

生成回應： Solve for x: 2x + 3 = 11.

$$\bigcup Q\left(\sum_{i=0}^{n-1} x\right) = Q\left(\sum_{i=0}^{n-1} k\right) = Q\left(\sum_{i=0}^{i-1} j\right)$$

$$\cup \backslash rlap\omega(X_k) = Q\left(\sum_{i=0}^{n-1} k\right)$$

==========

問題 4: Simplify the fraction 24/36.

生成回應： Simplify the fraction 24/36.

==========

問題 5: Find the derivative of f(x) = x^2 + 3x - 5.

生成回應： Find the derivative of f(x) = x^2 + 3x - 5.

==========

問題 6: Solve the quadratic equation: x^2 - 5x + 6 = 0.

生成回應： Solve the quadratic equation: x^2 - 5x + 6 = 0.

To find the solution of the quadratic equation x^2 - 5x + 6 = 0, substitute the value of x at 5 into the equation.

==========

問題 7: Evaluate the integral ∫0^1 x^3 dx.

生成回應： Evaluate the integral ∫0^1 x^3 dx. I think it is a polynomial. Any help is appreciated.

==========

問題 8: Find the eigenvalues of the matrix [[2,1],[1,2]].

生成回應： Find the eigenvalues of the matrix [[2,1],[1,2]].

$$〚\{array\}\{rcl\} ├[f(x)h┤] \& = \& \{\{dx \cdot y\} + ├[├| ├|\{┤ |x ├|2\}\}┤ |\^5\} ├|x ├|2. 〛 \{array\}]$$

The eigenvalues are the diagonal matrix elements.

==========

問題 9: Solve the differential equation dy/dx = y * tan(x) with y(0)=1.

生成回應： Solve the differential equation dy/dx = y * tan(x) with y(0)=1.

Solution 1.

Take the gradient of the function y(0) by setting dx = 0.

$$F(x) = -F(y)$$

==========

問題 10: Discuss the implications of the Riemann Hypothesis on the distribution of prime numbers.

生成回應： Discuss the implications of the Riemann Hypothesis on the distribution of prime numbers.

==========

經過整理，我們可以發現訓練過後的 GPT2 有以下兩種情況:

1. 生成錯誤的回覆(可能是答案錯誤，或者解答是亂碼)
2. 重複問題

種種跡象都表明，GPT2 並沒有在這次訓練中學習到回答以上數學題的能力，或者它似乎沒有真正理解這是一個「問題」，以至於生成出的內容並不符合我們的期待，亦顯示這絕對是一次糟糕的訓練。

五、 實驗結果檢討

根據實驗結果顯示:這是一次糟糕的訓練。我認為可以統整出以下三個問題。

第一個問題是，這次實驗的時間並不夠長，我沒有足夠的時間可以調整參數，才剛看完資料就必須開始實驗了，也來不及測試其他的模型是否可以勝任這些任務，這導致無法控制實驗的變量，亦沒有明顯的對照組，或許我可以在時間相對充裕的情況下進行實驗。

第二個問題是，餵給模型的資料過少，導致模型還沒學到足夠的資訊就被迫中止訓練，或許我可以在下次實驗的時候調大餵給模型的數據量。

第三個問題是，經費不足以提升設備效能，也導致這次實際訓練中時常模型還沒訓練完就被平台中止了，未來如果有機會完整實驗的話，我想經費會是一個很大的問題。

六、 未來展望

我還是希望可以訓練一個自己的模型出來，無論是未來工作上或大學學業上的使用，都有助於我完成被指定的任務，但首先我要解決以下問題:

- 訓練時間過短
- 訓練資料過少
- 經費不足

七、 心得與反思

雖然這次研究 LLM 的時間很短，但我還是學到了不少，以下是我學到的能力及其摘要:

能力	摘要
資料的整理能力	從原文論文中提取重要資料並用自己的言語描述
發現問題和解決問題	對程式 debug、解決實驗設計中隱含的問題
程式設計的能力	調用 API 和訓練模型
文書處理工具	利用不同的工具如 Latex 來描述實驗資訊

透過這次研究的經驗，讓我發現我還有很多需要學習的地方，從實驗設計的精確度到對資源的掌握度，這些都是需要靠我自己去完善的目標，而我也勢必要在進入大學前解決這些我固有的問題，才有助於我的未來發展。