作業 3 報告

Q1：Retriever 與 Reranker 微調

1. Retriever 訓練過程

訓練資料構成：

- 資料來源：data/train.txt
- 每筆資料包含：
  - rewrite：重寫後的查詢問題（query）
  - evidences：候選段落列表
  - retrieval_labels：0/1 標籤（1 表示正樣本、0 表示負樣本）
- 採樣方式：
  - 每個 query 對應 1 筆正樣本與約 4 筆負樣本
  - 每筆 query 形成 (query, positive, negative) 三元組

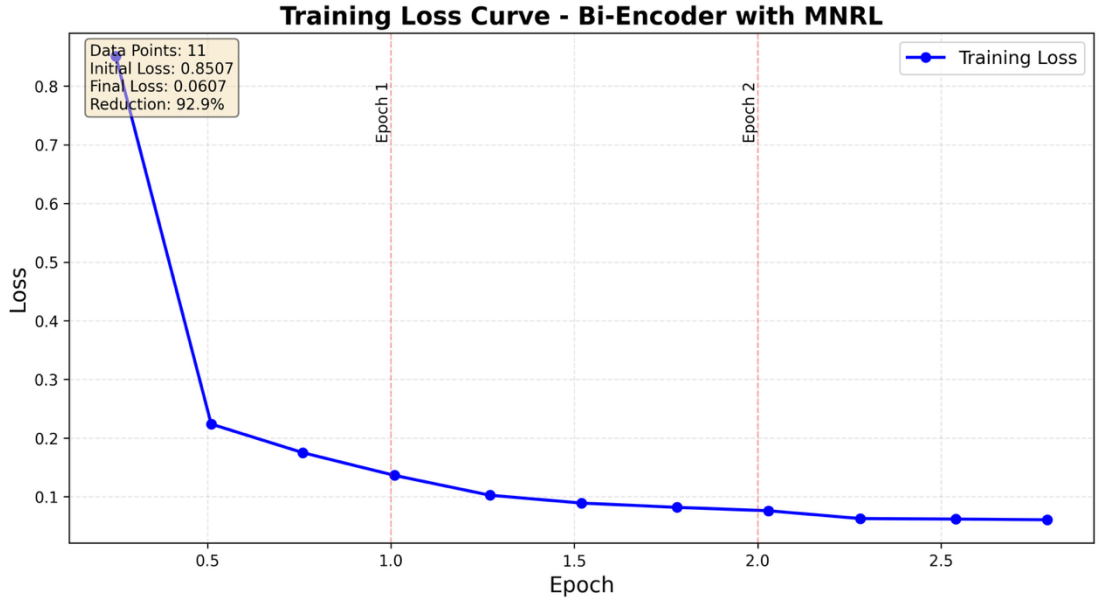
損失函數：

- Loss：MultipleNegativesRankingLoss (MNRL)
- 目標：讓正樣本的語意距離更接近、負樣本更遠
- 相似度計算使用 cosine similarity

超參數設定：

- model_name：intfloat/multilingual-e5-small
- batch_size：64
- epochs：3
- learning_rate：2e-5
- max_seq_length：512
- warmup_steps：500
- use_amp：True

訓練損失曲線：



損失由 0.85 穩定下降至約 0.06，收斂良好，表示語意對比學習成功。

後面發現後兩輪有一點點過擬合，最好的效果是第一輪

第一輪結束 Recall@10: 0.8677

第三輪結束 Recall@10: 0.8591

2. Reranker 訓練過程

訓練資料構成：

- 資料來源：train.txt
- 每筆 JSON 擴展為 (query, passage, label) pair
- 正樣本：retrieval_labels == 1
- 負樣本：retrieval_labels == 0
- 平均比例約 1:4

損失函數：

Loss：Binary Cross Entropy (BCE)

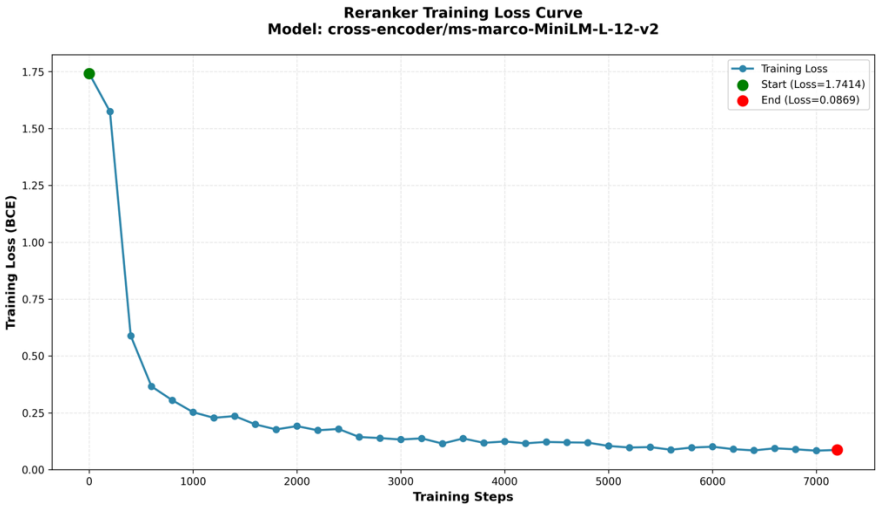
採用 pos_weight ≈ 4.0 平衡正負樣本比例

超參數設定：

- model_name：cross-encoder/ms-marco-MiniLM-L-12-v2
- learning_rate：1e-5
- batch_size：64

- epochs：2
- warmup_ratio：0.1
- pos_weight：4.0（正樣本加權）
- batch_sampler：Balanced（每個 batch 正負比例平衡）
- eval_strategy：每 200 steps 驗證一次（定期驗證避免過擬合）
- load_best_model_at_end：True

訓練與驗證損失曲線：



訓練損失由約 1.7 降至 0.08

驗證損失與訓練曲線一致，顯示良好收斂

3. 模型評估（停用 LLM）

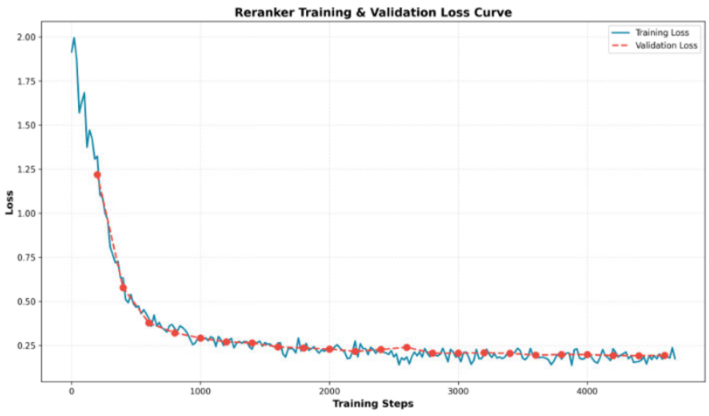
Retriever（第三輪的模型）：Recall@10 = 0.8591

用預訓練沒有微調的 Reranker 模型：MRR@10 = 0.7558（表現良好）

微調後 Reranker 模型：MRR@10 = 0.2011

說明：第二個 Reranker 模型懷疑過擬合，之後重新訓練，加入 batch sampler 與驗證損失監控、以及調整學習率後，雖然 loss 圖看起來一樣很好，但還是過擬合，最好的成績是訓練前 1000 步的模型，分數最高是 0.5347，但還是比起預訓練模型表現較差，因此評估預訓練模型表現較好。

```
============================================================
📊 評估結果
============================================================
Queries evaluated: 3342
Recall@10: 0.8677 (86.77%)
MRR@10 (after rerank): 0.5347
Bi-Encoder CosSim: N/A (LLM disabled)
```



Q2：Prompt 優化

設計流程：

- 先使用 claude sonnet 4.5 提供的版本測試後再調整
- 一開始先使用設定角色、規則的方式寫

Prompt 實驗表：

v1：設定角色、行為規則，要求簡潔直接的答案

- System Prompt 設計
  - 角色定義：設定 LLM 為 "precise question-answering assistant"
  - 核心約束：強調只能基於提供的段落回答（"based ONLY on the given context"）
  - 四大規則：
  - 禁止使用外部知識
  - 要求簡潔回答
  - 無法回答時返回 "CANNOTANSWER"
  - 禁止加入 "Based on the context" 等前綴
- User Prompt 設計
  - 結構化呈現：用 [Passage 1], [Passage 2] 編號段落
  - 明確標示：標注段落已經過 reranker 排序
  - 重複指令：在 user prompt 再次強調規則
- Answer Parsing 設計
  - 移除常見前綴（"The answer is:", "Based on the context"）
  - 提取 "Answer:" 後的內容
  - 處理 CANNOTANSWER 特殊情況
  - 移除引號和多餘空白
- 成績
  - Recall@10: 0.8677 (86.77%)
  - MRR@10 (after rerank): 0.7669
  - Bi-Encoder CosSim: 0.2510

v2：只設非常簡單的提示詞看效果，想說會不會是提示詞帶歪

```python
from typing import List
import re

def get_inference_system_prompt() -> str:
    """簡化版系統提示詞"""
    return "Answer the question using only the given context. If not found, reply CANNOTANSWER."


def get_inference_user_prompt(query: str, context_list: List[str]) -> str:
    """簡化版使用者提示詞"""
    context_text = "\n\n".join([f"[{i+1}] {ctx}" for i, ctx in enumerate(context_list)])
    return f"Context:\n{context_text}\n\nQuestion: {query}\n\nAnswer:"


def parse_generated_answer(pred_ans: str) -> str:
    """簡化版答案解析"""
    if not pred_ans:
        return "CANNOTANSWER"

    parsed = pred_ans.strip()

    if "CANNOTANSWER" in parsed.upper():
        return "CANNOTANSWER"

    return parsed if parsed else "CANNOTANSWER"
```

- 成績
    - Recall@10: 0.8677 (86.77%)
    - MRR@10 (after rerank): 0.7669
    - Bi-Encoder CosSim: 0.2510

發現成績跟原本完全一模一樣，不太合理，再繼續用小批次測試找問題，懷疑是過濾函式的問題

v3：對照 result.json 找生成的答案比對，針對生成內容與正確答案重寫 parse_generated_answer( ) 過濾函式，也發現有時候 LLM 會針對每個 passage 都各自回答，所以有強調只需要一個答案。

- 測試檔全部測試的成績
    - Recall@10: 0.8677 (86.77%)
    - MRR@10 (after rerank): 0.7669
    - Bi-Encoder CosSim: 0.3857

結論：

V3 效果最好！

不過發現改變幾個字就會影響結果，所以提示詞有點懸！

有些我看起來提示更好的，但好像效果更差。

```python
def get_inference_system_prompt() -> str:
    """第四版：更嚴格的答案要求"""
    return (
        "You are a precise question-answering assistant. "
        "You must answer directly and accurately based on the provided passages."
    )


def get_inference_user_prompt(query: str, context_list: List[str]) -> str:
    """第四版：強調必須完全引用原文，不得改寫"""
    # 上下文段落編號
    context_text = "\n\n".join([f"[{i+1}] {ctx}" for i, ctx in enumerate(context_list)])

    return (
        f"Context passages:\n{context_text}\n\n"
        f"Question: {query}\n\n"
        f"Instructions:\n"
        f"1. Read all passages carefully to find the only answer\n"
        f"2. Your answer MUST be copied EXACTLY from the passage text - do NOT paraphrase or change any words\n"
        f"3. Copy the relevant sentence(s) word-for-word from the passage\n"
        f"4. If the answer is not found in any passage, write exactly: CANNOTANSWER\n\n"
        f"Answer:"
    )


def parse_generated_answer(pred_ans: str) -> str:
    """解析模型生成的答案，提取 assistant\\n<think>\\n\\n</think>\\n\\n 後面的內容"""

    # 方法1: 尋找 </think> 後的內容
    think_pattern = r'</think>\s*\n\s*(.+?)(?:\n|$)'
    match = re.search(think_pattern, pred_ans, re.DOTALL)
```

Q3：其他分析（2%）

計畫分析方向：

1. 比較 Reranker 模型是否能明顯提升 MRR
2. 增加輸入的資料筆數可不可以解決這問題，效果是否可以打平有 Reranker 模型

測試方式：

1. 先測原本沒變時的成績，都用 100 筆資料
2. 測不用 reranker 模型，一樣用 Top-3 的成績看結果
3. 決定要是 passage 送入比 3 筆多，結果會不會更好，跟用 Reranker 一樣

(提示詞和過濾答案都是用同一個版本，上方第二題測試最好的 v3)

測試：

1. 測試 100 筆資料的結果，用微調後的 retriever 模型和 reranker 預訓練模型
   - 成績
     - Recall@10: 0.8900
     - MRR@10 (after rerank): **0.7745**
     - Bi-Encoder CosSim: 0.4143

2. 測試不使用 reranker，直接用 retriever 模型的排名，同樣 Top-3
   - 成績
     - Recall@10: 0.8900
     - MRR@10: 0.6633
     - Bi-Encoder CosSim: 0.4026

看起來沒有重新排名，正確答案的排名真的有變差一點點，而 LLM產生答案的分數反而差 0.01，似乎沒有差探多

3. 測試不使用 reranker，不過是 Top-5，看 LLM 結果會不會變化

- 成績
  - Recall@10: 0.8900
  - MRR@10: 0.6633
  - Bi-Encoder CosSim: **0.4131**

Bi-Encoder CosSim 成績已經追上第一個有用 Reranker 的版本了，**看起來只要讓文章數量夠多，就可以彌補排名上的缺點**，可能是因為送進 LLM 的資訊其實是不分順序的，都會讀一遍，所以只要有出現那就會讀到，當然我覺得提示詞也很重要，好的提示詞才可以讓 LLM 不會亂生成答案

4. 由於 Top-5 表現很好，於是測試看看 Top-8，會因為送進的參考文章變多而更準還是更不準
   - 成績
     - Recall@10: 0.8900
     - MRR@10: 0.6633
     - Bi-Encoder CosSim: **0.4039**

Top-8 的版本成績下降了，不過最終成績還是比 Top-3 版本好，或許雖然更多文章會影響判斷，但太少文章 LLM 就連參考的機會都沒有

結論：

不用 reranker 模型，正確答案的排名會變差一些，不過利用 Retriever 模型比對文章的排名，從 Top-3 增加至 Top-5，可以幾乎彌補掉這段差異，但假如增加更多參考資料，Top-8 結果就比 Top-5 差了，但還是比 Top-3 更好。
假如有好的提示詞，且只用 Retriever 模型的狀況，算力效能可以省很多，那用 Retriever 模型 Top-5 是一個很好的決定！

附錄：

硬體環境：RTX 3090 (24GB VRAM)

參考訓練程式：

Retriever:
https://github.com/huggingface/sentence-transformers/blob/master/examples/sentence_transformer/training/ms_marco/train_bi-encoder_mnrl.py

Reranker:
https://github.com/huggingface/sentence-transformers/blob/master/examples/cross_encoder/training/ms_marco/training_ms_marco_bce.py