2024 ADL HW1 Report

r12944005 陳乙馨

Q1: Data processing

Tokenizer

NLP 模型無法直接處理字詞，需要先將它們轉換為 token進行處理，當文本經過 tokenization 後，會包含以下主要資訊：

- Token IDs : 每個 token 都會轉換成一個對應的數字 ID，這些 ID 來自模型的詞彙表。
- Attention mask : 用來區分 token 和 padding。有效 token 會被標記為 1，padding token（補齊固定長度）則會被標記為 0。
- Offsets（Offset Mapping）: Offsets 是 token 和原始文本之間的位置 mapping。每個 token 都會對應文本中的起始和結束位置。
- Special Tokens : 在文本中添加的特殊標誌，例如 `[CLS]`（用於表示整個句子的輸入）和 `[SEP]`（用於分隔句子或段落）。
- Squence IDs : 用以判斷該 token 是屬於文本輸入的哪一部分 (Ex : 上下文或是問題)

在這個作業中，我分別使用了 `google-bert/bert-base-chinese` 和 `hfl/chinese-lert-base` 兩種模型及其對應的 tokenizer，他們皆是使用了 WordPiece 演算法，WordPiece 會把每個漢字作為一個最基本的 token 單位、再根據出現的頻率高低決定要不要合併某些子詞，頻率出現較低的字對將會優先被合併，這樣可以更有效地減少詞彙表中的低頻詞 (與 BPE 中頻率高的子詞會先被拆解掉不同)，同時優化模型對新詞的處理能力。

Answer Span

How did you convert the answer span start/end position on characters to position on tokens after BERT tokenization?

我直接使用了 hugginggace 上 sample code 的方法，首先先將原始文本的 ground truth character span `start_char` 和 `end_char` 記錄起來，再利用 offset mapping 去尋找對應到 `start_char` 的那個 token 的 index、以及對應到 `end_char` 的那個 token 的 index，即完成將 character span 轉換為 token span 的標記。

如果碰到一些例外狀況，例如 : 如果涵蓋整段上下文的起始 token 與結尾 token map 回原始文本後並沒有涵蓋到 `(start_char, end_char)`，則直接插入 `cls_index` 代表沒有答案。

After your model predicts the probability of answer span start/end position, what rules did you apply to determine the final start/end position?

對於每個問題，模型會 output `start_logits` 和 `end_logits`，他們分別是一個 array，經過 softmax 後記錄每個 token 是 span 的起點或是終點的機率分布 (Ex : 當 `start_logits = [0.1, 0.5, 0.2, 0.3, 0.1]` ，則代表第 1 個 token 被預測為答案開始位置的概率為 0.1、第 2 個 token 的概率為 0.5 ... 依此類推)。但是因為每個問題的 tokens 長度不一、array 的長度也會不同，因此需要透過 function `create_and_fill_np_array` 做 padding 使所有 `start_logits` 和 `end_logits` 長度一致、方便做後續處理。

我篩選最終答案的做法也是參考 huggingface 上的 sample code，就是在 `start_logits`、`end_logits` array 中取作為起點概率 + 終點概率最高的 span 為答案，最後再把 token map 回文本，不過需要注意排除一些例外像是 token index 不在 offset mapping 的範圍內、導致無法 map 到原始的文本。

Q2 : Modeling with BERTs and their variants

- Optimizer : AdamW
- Scheduler : Linear
- Loss function : Cross-Entropy Loss (According to huggingface document)
- max_seq_length : 512

Basic Result

	Multiple Choice	Question Answering
model	`google-bert/bert-base-chinese`	`google-bert/bert-base-chinese`
epochs	3	10
learning rate	3e-5	1e-5
per_device_train_batch_size	1	2
gradient_accumulation_steps	2	4
evaluation accuracy	0.9541	0.7913
Kaggle public score		**0.6678**

My Best Result (Variant)

	Multiple Choice	Question Answering
model	`hfl/chinese-lert-base`	`hfl/chinese-lert-base`
epochs	3	10
learning rate	3e-5	1e-5
per_device_train_batch_size	2	2
gradient_accumulation_steps	4	4
evaluation accuracy	0.9684	0.8311
Kaggle public score		**0.8035**

我原本是使用 spec 上過基本 baseline 的參數以及模型 (即 basic result)，結果表現不彰，因此我做了幾項改變。

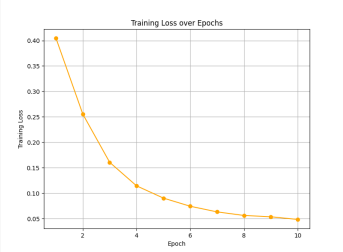
1. BERT to LERT： 根據 作者 的 github，LERT在十個中文自然語言理解任務中表現都遠優於 baseline，因為 LERT 不僅依賴掩碼語言模型（MLM）進行預訓練，還引入了三種語言學任務，這使得模型比起 bert 能夠更深入地學習語言的結構和特徵。
2. 提高 Multiple Choice 的 batch size（2 到 8），增強收斂性。

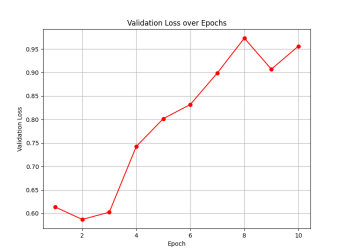
以下是更換模型以及調整參數後的結果比較：

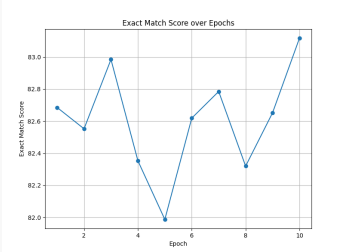
	Multiple Choice Accuracy	Question Answering Accuracy	Kaggle Public Score
`google-bert/bert-base-chinese`	0.9541	0.7913	0.6678
`hfl/chinese-lert-base`	0.9684	0.8311	0.8035

Q3 : Curves

以下是根據 my best result 所繪製的曲線圖。







Q4 : Pre-trained vs Not Pre-trained

Scratch Result

	Multiple Choice	Question Answering
tokenizer	`google-bert/bert-base-chinese`	`google-bert/bert-base-chinese`
epochs	3	10
learning rate	3e-5	1e-5
per_device_train_batch_size	2	2
gradient_accumulation_steps	4	4
evaluation accuracy	0.5194	0.0708

我使用了跟 my best result 完全相同的參數、以及 `bert-base-chinese` 的 tokenizer 來訓練 scratch model，結果顯示他在 multiple choice 的 task 上仍有超過五成的準確度，但對於需要精確得到 span 的 QA 任務來說，準確度就只剩下不到一成。

以下是他和 pre-trained `bert-base-chinese` 的比較：

	Multiple Choice Accuracy	Question Answering Accuracy
Pretrained	0.9541	0.7913
Not-pretrained	0.5194	0.0708

Reference

- ChatGPT (mainly for sample code comprehension and plotting curves)
- WordPiece Tokenization
- LERT Introduction
- Hugging Face : Multiple Choice
- Hugging Face : Question Answering
- HuggingFace : Tokenizer