ADL Homework 1 Report

B10705016

Q1: Data processing (2%)

Tokenizer (1%)

Describe in detail about the tokenization algorithm you use. You need to explain what the algorithm does in your own ways. Just answering with "I called the () function" is not allowed.

在我的專案中, 我使用了 pre-trained model hfl/chinese-roberta-wwm-ext 的 tokenizer, 這個 tokenizer 是專門為中文文本設計的, 採用了 WordPiece tokenization 演算法。這樣的選擇不僅能有效處理中文的特性, 還能提高模型的理解與生成能力。

WordPiece Tokenization 演算法

- 詞彙建構:這個 tokenizer 一開始會有一個基本的詞彙庫, 裡面包含了語言中的所有獨立字元。它會不斷地透過合併最常見的符號(無論是字元還是子詞)來擴展詞彙庫, 這個過程是基於在大量語料庫中出現的頻率進行的。對於中文來說, 這通常會產生一個包含常用字、詞彙和有意義的子詞的詞彙庫。
- 分詞過程
    - 第一步:輸入文本處理
      輸入的文本會先進行標準化(例如:轉成小寫、處理標點符號)。由於中文的單詞之間並沒有空格, 所以文本會被視為一連串連續的字元。
    - 第二步:貪婪式最長匹配優先分詞
      從文本的開頭開始, tokenizer 會試著匹配詞彙庫中存在的最長子串。它會採用一種貪婪的方式, 總是選擇最長的匹配子詞。
    - 第三步:子詞切分
      如果在詞彙庫中找到一個子串, 它就會被添加為一個 token, 然後 tokenizer 會根據該子串的長度向前移動。如果沒有找到匹配的子串, tokenizer 就會把字元拆分成更小的單位, 必要時會使用 [UNK](未知) token。
- 特殊 Tokens 與標記
    - 延續子詞:在一個詞中但不在開頭的子詞通常會加上 ## 前綴, 以表示它們是延續部分。在中文分詞中, 由於語言的特性, 這種情況可能不那麼常見。
    - 特殊 Tokens:
        - [CLS]:這個 token 會在序列的開頭加上, 代表分類的標記。
        - [SEP]:用來分隔不同的序列(例如:問題與上下文之間)。
- 範例
    - 輸入文本: "我喜歡自然語言處理"
    - 分詞步驟:
        - tokenizer 會尋找最長的子詞:"我"、"喜歡"、"自然語言處理"
        - 產生的 Tokens: ["我", "喜歡", "自然語言處理"]

- 算法優點

    - 處理沒看過的詞彙: 透過將詞彙拆分為子詞, tokenizer 可以處理那些在詞彙庫中不存在的稀有或新詞。
    - 效率: 在詞彙大小與表達大量可能單詞的需求之間達到平衡。

Answer Span (1%):

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

將字元位置轉換為 Token 位置

- Tokenization 與位移:在對上下文和問題進行 tokenization 時, 我使用了帶有 return_offsets_mapping=True 參數的 tokenizer。這樣做可以為每個 token 返回一個 offset_mapping, 這是一個元組的列表, 顯示了每個 token 在原始文本中的起始和結束字元位置。
- 確定答案範圍的 Tokens
    - 答案字元位置:在資料集中的每個答案都有一段文本和一個在上下文中的 answer_start 位置。answer_end 位置是通過 answer_start + len(answer_text) 計算得出的。
    - 映射到 Tokens:我會迭代 offset_mapping, 找出包涵答案範圍的 tokens。token_start_index 是第一個 token, offset_mapping[token_start_index][0] >= answer_start。而 token_end_index 則是最後一個 token, offset_mapping[token_end_index][1] <= answer_end。
- 處理邊界情況
    - 上下文截斷:如果上下文太長而被截斷, 答案可能就不會出現在 tokenized 的輸入中。在這種情況下, 我會將 token_start_index 和 token_end_index 設定為 [CLS] 的索引。這樣做可以避免因為上下文的截斷而造成的錯誤映射, 確保模型能正確地處理不同情況。
    - 無效範圍:如果計算出的 token 索引不能形成有效的範圍, 我會將其預設為 [CLS] token 的索引。確保在遇到無效範圍的情況下, 模型依然能夠保持穩定的行為, 避免因為範圍問題而導致的錯誤。

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

確定模型預測後的最終起始/結束位置

- 模型輸出:模型會輸出 start_logits 和 end_logits, 這兩個指標提供了每個 token 作為答案範圍的起始或結束位置的概率。
- 選擇最佳範圍的規則
    - 規則 1:有效範圍
      選擇生成所有可能的範圍, 其中 start_index <= end_index, 並且範圍的長度不超過 max_answer_length。
    - 規則 2:分數計算
      對於每個有效範圍, 計算分數, 方法是將 start_logits[start_index] 和 end_logits[end_index] 相加。
    - 規則 3:選擇最佳範圍
      選擇分數最高的範圍作為最終預測的答案範圍。

- 後處理
    - 位移映射：使用 offset_mapping 將 token 索引轉換回原始上下文中的字元位置。
    - 提取答案文本：根據預測範圍的字元位置，從上下文中提取對應的子字串。

Q2: Modeling with BERTs and their variants (4%)

Describe (2%)

Your model.

- Multiple Choice 模型
    - 架構：基於一個預訓練的 transformer 架構，專門用於理解語言和上下文。具體來說，它經過調整，以處理多選任務，目標是在一組可能的答案中選擇正確的選項。
    - 輸入處理
        - 輸入組合：對於每個問題，模型會通過將問題與每個可能的段落（選項）結合，來生成一組輸入組合。
        - 編碼：每個問題-段落對會使用 transformer 的層進行編碼，以捕捉它們之間的語義關係。
    - 分類層：編碼後，模型會將表示傳遞到一個分類層，該層為每個選項輸出一個分數。預測時，會選擇分數最高的選項作為模型對最相關段落的預測。
    - 使用的預訓練模型：hfl/chinese-roberta-wwm-ext，一個在大型中文語料庫上預訓練的 RoBERTa 模型，使用了 Whole Word Masking（WWM）。WWM 確保在訓練過程中整個單詞被遮蔽，而不是子詞，這有助於模型更好地理解中文的詞級語義。

- Span Extraction 模型
    - 架構：基於一個預訓練的 transformer 架構，專門為問題回答任務設計，答案是在給定上下文中的一段文本。
    - 輸入處理
        - 問題與上下文組合：模型接收一個問題和一個上下文段落作為輸入。
        - 編碼：兩個輸入會被串接在一起，然後通過 transformer 的層進行處理，以生成豐富的組合表示。
    - 輸出層
        - 起始和結束位置預測：模型有兩個輸出層，分別預測每個 token 作為答案範圍的起始或結束位置的概率。
        - Softmax 層：這些層會應用 softmax 函數，生成所有 token 位置的概率分布，以分別預測起始和結束位置。
    - 答案提：模型會選擇概率最高的 token 位置作為預測的答案起始和結束位置。利用這些位置，從上下文中提取出準確的答案文本。
    - 使用的預訓練模型
        - 模型名稱：hfl/chinese-roberta-wwm-ext，敘述如上段。

The performance of your model.

我 fine-tune 的 mutliple-choice model 在第三個 epoch 時，能在 validation set 取得約 96% 的 accuracy 。

```
epoch 0: {'accuracy': 0.9561316051844466}
]epoch 1: {'accuracy': 0.9328680624792289}
/s]epoch 2: {'accuracy': 0.9587902957793287}
```

我 fine-tune 的 span-extraction 在第三個 epoch 時，能在 validation set 取得約 82.4 的 f1 score。

```
{'exact_match': 82.38617480890662, 'f1': 82.38617480890662}
```

最後，透過串接兩個模型，解決作業一的 task，在 kaggle competition 上的 public score 是 0.79、private score 則是 0.8。

Private Score ⓘ	Public Score ⓘ
0.80044	**0.78947**

The loss function you used.

- Multiple-Choice 模型：Cross-Entropy Loss 來比較每個選項的預測概率與真實標籤。
- Span-Extraction 模型：使用了兩個 Cross-Entropy Loss 的總和，分別針對起始位置和結束位置進行計算。

The optimization algorithm (e.g. Adam), learning rate and batch size.

兩個模型的 optimizer 及 hyperparameters 如下：

- Optimizer: AdamW (adaptive lr + weight decay, helps model converge faster)
- Learning rate: 3e-5, or 0.00003
- Batch size: 4
- Max sequence length: 512
- Epochs trained: 3

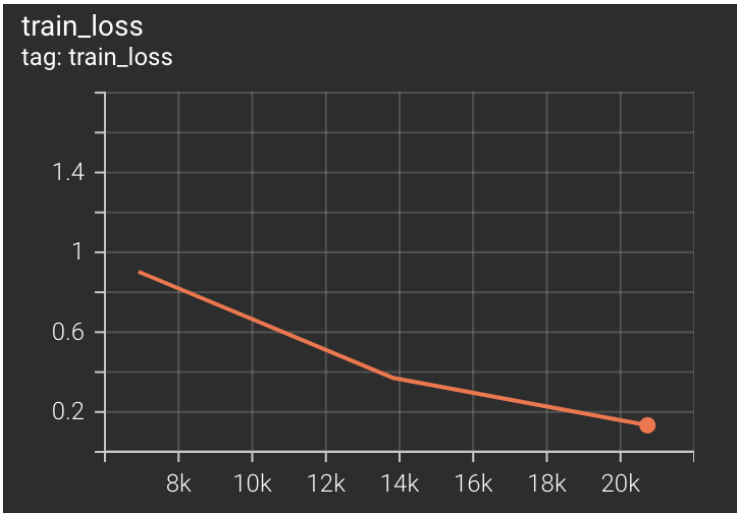
Try another type of pre-trained LMs and describe (2%)

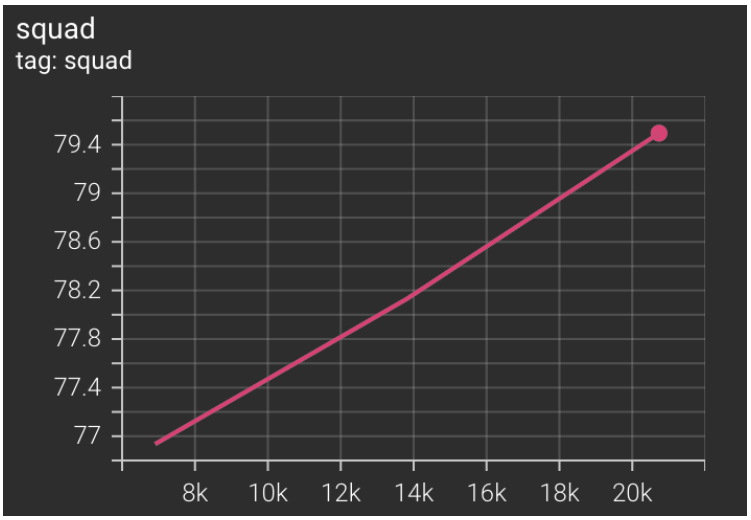
The new model.

我選擇了 Google 的 Bert-based Chinese 作為另一個預訓練模型。

The performance of this model.

Google 的 Bert-based Chinese 模型在訓練 3 個 epochs 後，exact match 分數達到 79.49，比原本選用的 hfl/chinese-roberta-wwm-ext 差了一些，但我想應該仍能壓線通過 kaggle strong baseline。此外，一個有趣的觀察是折線圖的線型相較漂亮，每個 epoch 都能穩定增加超過 1 分。





The difference between pre-trained LMs (architecture, pretraining loss, etc.)

- 相同處
  - 架構：兩個模型（Google BERT-Base Chinese 和 hfl/chinese-roberta-wwm-ext）都使用相同的架構，各有 12 層的 Transformer encoder，hidden layer 大小為 768，還有 12 個 attention heads，總參數數量約為 1.1 億。
  - 模型大小和計算資源：這兩款模型的大小都差不多，所以在 fine-tuning 和推論的時候，所需的計算資源也相對接近。
- 相異處
  - 預訓練 loss function 與策略：BERT-Base Chinese 採用的是字符級的掩蔽（Masked Language Modeling, MLM）和「下一句預測」(Next Sentence Prediction, NSP)，而 hfl/chinese-roberta-wwm-ext 則使用整詞掩蔽（Whole Word Masking, WWM），並省略了 NSP，專注於 MLM，這樣的改變使得模型在理解上下文時更有優勢。

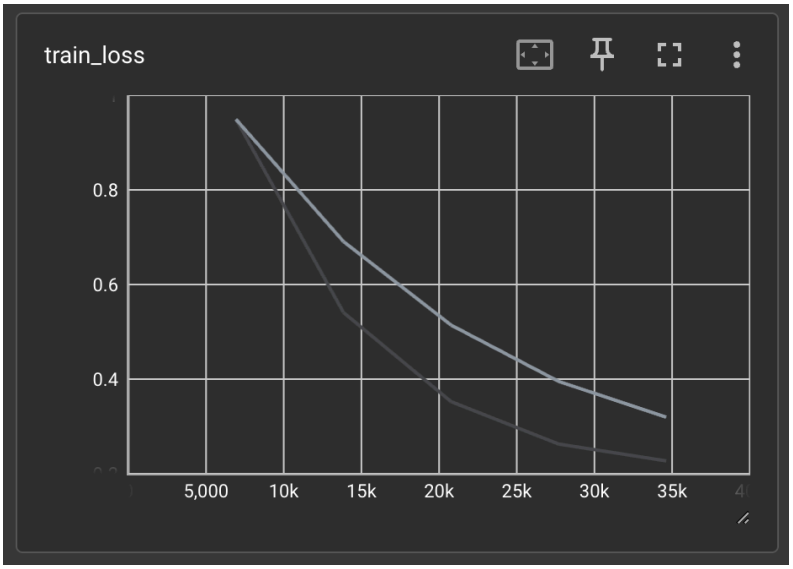
- 掩蔽策略：在 BERT 裡，掩蔽是隨機挑選單個字符，而 hfl 模型則會同時掩蔽整個單詞，這樣能夠更好地把握語詞的意義，感覺就像是從字面上進入詞彙的深層理解。
- 預訓練語料庫：hfl 模型訓練使用了擴展版的語料庫，涵蓋的資料來源更廣泛，因此在多樣性上相較於 BERT 有明顯的優勢。
- 訓練技術與優化：hfl 模型運用了 RoBERTa 的訓練策略，包括動態掩蔽、更大的批量大小和調整學習率等，這些技術上的優化讓模型在訓練過程中更容易收斂，性能自然也更出色。
- 下游任務性能：在多個標準測試中，hfl 模型的表現通常都超越了 BERT-Base Chinese，特別是在需要深入理解語境的任務上，像是閱讀理解（reading comprehension）和命名實體識別（named entity recognition, NER）。

Q3: Curves (1%)

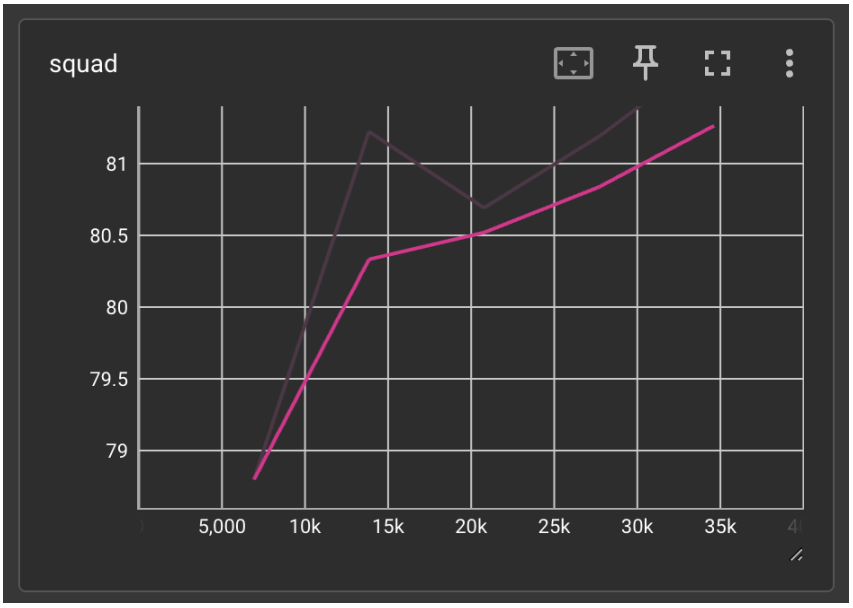
Plot the learning curve of your span selection (extractive QA) model. You can plot both the training and validation set or only one set. Please make sure there are at least 5 data points in each curve.

我使用 tensorboard 將 span selection model 訓練 5 個 epochs 過程中，每一個 epoch 結束後的 train loss 和 exact match 分數以折線圖呈現。X 軸單位是 steps，所以 scale 看起來很大，但實際每個 epoch 結束後才會用 validation set eval 並記錄一次，因此圖表中只有 5 個資料點。

Learning curve of the loss value (0.5%)



Learning curve of the Exact Match metric value (0.5%)



Q4: Pre-trained vs Not Pre-trained (2%)

Train a transformer-based model (you can choose either paragraph selection or span selection) from scratch (i.e. without pretrained weights).

Describe the configuration of the model and how you train this model (e.g., hyper-parameters).

利用同樣的訓練程式碼, 只要用以下參數執行, 就可以從零訓練模型。

```
!python multiple_choice_train.py \
 --train_file train_data.json \
 --validation_file valid_data.json \
 --model_name_or_path hfl/chinese-roberta-wwm-ext \
 --tokenizer_name hfl/chinese-roberta-wwm-ext \
 --output_dir ./output \
 --per_device_train_batch_size 4 \
 --per_device_eval_batch_size 4 \
 --learning_rate 3e-5 \
 --num_train_epochs 3 \
 --max_seq_length 512 \
 --pad_to_max_length \
 --lr_scheduler_type cosine \
 --seed 42
```

我用了 transformers 套件初始化新的模型(tokenizer 仍用 hfl 的預訓練模型), hyperparameters 如下:

- vocab size: 30522
- hidden layer size: 256
- number of hidden layers: 4
- number of attention heads: 4
- feed forward layer size: 1024
- activation function: gelu
- hidden layer dropout rate: 0.1
- attention layer dropout rate: 0.1, # Dropout probability for attention layers
- max sequence length: 512, # Maximum sequence length

Describe the performance of this model v.s. BERT.

自己從零訓練的模型, 在第十個 epochs 時仍只能達到 6.4 的 exact match score。訓練這個小模型 10 個 epochs 與訓練 pre-trained 的模型 2 個 epochs 花費約一樣的時間, 但成效相差甚遠。

```
]
]
 Evaluation metrics: {'exact_match': 6.4473246925889, 'f1': 6.4473246925889}
/s]
```

不過可以發現, 從零開始訓練的模型每個 epoch 之間進步的幅度是顯著的, 在我的實驗中, 每個 epoch 約會進步 0.6 以上的 exact_match score, 而 fine-tune 則沒有如此大的進步幅度。這是因為 pre-trained 模型已經事先利用超大語料庫幫我們教會模型中文語言的基礎理解, 我們只需要將其的理解透過微調轉化成更適合解決任務的格式。而從零開始訓練則有點像是同時教模型理解語言和解決任務。

因此, 使用 pre-trained models 來 fine-tube 能夠大幅度的減少訓練時間。如果以人類做比喻, 有點像是科技公司比起還不會講話的寶寶, 更願意招攬有工程背景的畢業生, 因為即使剛畢業的大學生還沒有實務經驗, 但在扎實的工程背景下, 只需要很少的教育及訓練, 就可以很快適應工作。

如今要用深度學習方法來解決一個問題, 網路上如此多開源的 pre-trained models 選項, 是一件多麼幸福的事！