NLP Assignment 2 – Question Answering

109504502 資電三 陳紓嫻

I. Data preporcessing

A. 文本處理

將文本讀入後，根據『|||』先將 question、snippets、answer 做分類，並使用 regex 將 snippets 分段，分句之後發現訓練集資料的每筆 snippets 大約落在 41~51 之間，最後存入 QA structure。

```
for line in file.readlines():
    snippets, question, answer = line.split('|||')
    question = question[1:-1]
    answer = answer.split('\n')[0][1:]
    contexts =
    [token[4:-5] for token in re.findall(r"<s>[^<>]*<\/s>", snippets)]
    examples.append(QA(question= question,
                       ans= answer,
                       snippets= contexts)
```

B. 訓練文本選擇

由於每筆訓練資料的 snippets 都很多，無法全部放進去預測，因此採用 BM25 演算法，挑選出最有可能產生出答案的文本，但並不是因為所有的 snippets 都包含答案，因此使用 BM25 也有可能會選到沒有答案的 snippet，所以在挑選訓練集的部分採用以下方法：

i. 利用 BM25 挑選前 15 名有可能存在答案之文本

ii. 尋找每筆資料的答案 index，有無答案的 snippet 都先分別紀錄

iii. 由於做預測時，可能會有無答案的文本，因此對於有無答案的測資都各挑選一筆作為訓練資料

iv. 因為只挑 15 筆資料做選擇，所以最終有可能會有幾筆訓練資料只有『無答案』的測資，無答案的起始結束 index 最終設為(0,0)

v. 在文本挑選時，同時計算 snippet 中 answer 的 index

vi. 全部的處理好的資料都儲存於『字典』結構中，方便後續操作

```
def BM25(datas: list, is_train: bool = True):
    article_list, ans_snippets = [], []
    for (i, data) in enumerate(datas):
        querys = data.question
        snippets = data.snippets
        ans = data.ans
        article_list.clear()
        for a in snippets:
            stemmed_tokens = word_tokenize(a)
            article_list.append(stemmed_tokens)
        # bm25模型
        bm25Model = bm25.BM25(article_list)
        average_idf = sum(map(lambda k: float(bm25Model.idf[k]), bm25Model.idf.keys())) / len(bm25Model.idf.keys())
        scores = bm25Model.get_scores(querys, average_idf)
        a = torch.tensor(scores)

        v, idx = a.topk(k=15, largest=True)
        choose 1 snippet that has answer and 1 snippet that has no answer to ans_snippets
        find the answer index in ans_snippets
```

C. 訓練資料結構轉換

使用 Dataset 函式建置資料型態，主要分成兩個部分：

1. 初始化 Dataset → __init__
   i. 在這邊使用 transformer 的 tokenizer，由於需要 tokenizer 回傳 offset_mapping，因此採用 BertTokenizerFast 的函式(不知道為啥 BertTokenizer 會報錯)，並且將問題和文本合併在一起，最終訓練時會使用以下資訊：

訓練模型	評估模型
a. input_ids	a. offset_mapping
b. token_type_ids	b. ans
c. attention_mask	c. snippet
d. start_positions	
e. end_positions	
(已對應為 token 的 index)	

2. index 轉換：
   i. 使用 transformer 內建的 char_to_token 函式，將原本 answer 的 index 轉換為 token 的位置
   ii. 如果文本太長被截斷導致答案找不到，一樣會將 token 的 index 設為(0,0)
      a. 最後將此資訊更新到訓練資訊裡
         ✓ start_positions
         ✓ end_positions

```
def add_token_positions(self):
    # 初始化列表以包含答案start/end的標記索引
    start_positions = []
    end_positions = []
    for i in range(self.len):
        if self.start_idx[i] == -1 or self.end_idx[i] == -1:
            start_positions.append(0)
            end_positions.append(0)
            continue

        # 使用char_to_token方法追加開始/結束標記位置
        start_positions.append(self.encodings.char_to_token(i, self.start_idx[i], sequence_index=1))
        end_positions.append(self.encodings.char_to_token(i, self.end_idx[i], sequence_index=1))

        # 如果起始位置為None，則答案已被截斷
        if start_positions[-1] is None:
            start_positions[-1] = 0
        # end position無法找到，char_to_token找到了空格，所以移動位置直到找到為止
        shift = 1
        while end_positions[-1] is None and self.end_idx[i] - shift > 0:
            end_positions[-1] = self.encodings.char_to_token(i, self.end_idx[i] - shift, sequence_index=1)
            shift += 1
        if end_positions[-1] is None:
            end_positions[-1] = 0
    # 用新的基於標識的開始/結束位置更新我們的encodings對象
    self.encodings.update({'start_positions': start_positions, 'end_positions': end_positions})
```

II. Model structure

```
self.hidden_size = hidden_size

# BERT
self.bert = BertModel.from_pretrained(MODEL_NAME)

# Linear Layer
self.qa_logits = nn.Linear(hidden_size, 2)  # 2 for start and end logits
```

本次作業的模型主要使用 **BERT** 架構為主體，主要分層兩層

A. Bert ：

這裡採用 deepset/bert-base-uncased-squad2 的預訓練模型，整體與 bert-base-uncased 的架構一樣

B. Linear Layer ：

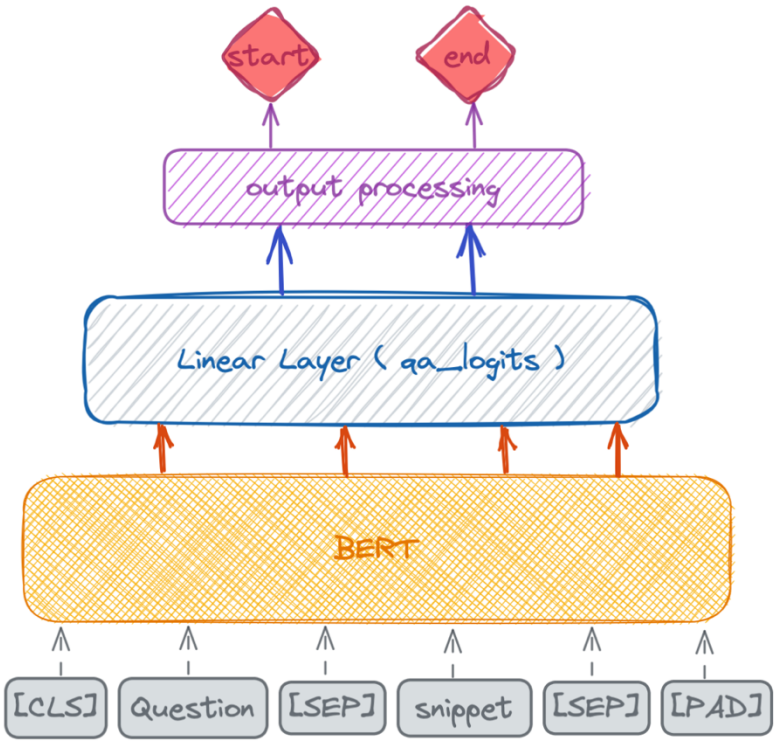
output_size = seq_len * 2　# (start, end)

C. 損失值在 forward 裡處理，輸出後要另外處理資料

```
if start_positions is not None and end_positions is not None: # do training
    ignored_index = start_logits.size(1)
    # clamp: limit the value between 0 and ignored_index
    start_positions = start_positions.clamp(0, ignored_index)
    end_positions = end_positions.clamp(0, ignored_index)

    loss_fn = nn.CrossEntropyLoss(ignore_index=ignored_index)
    start_loss = loss_fn(start_logits, start_positions)
    end_loss = loss_fn(end_logits, end_positions)
    total_loss = (start_loss + end_loss) / 2
    output = (total_loss,) + output
```

D. 模型結構圖 ：



III. Training process

一共跑五個 epoch，在之中選擇 f1-score 最高的模型，儲存起來，每個 epoch 跑完後會測試 dev 資料集查看訓練整體狀況，在這邊我們利用 tokenizer 完後產生的 offset_mapping 來將預測出來的 token index 比對回去原本 snippet 的字串，利用 eeclass 給的 f1-score 函式做計算，exact_match 為有答案之測試集的分數，all_match 為有無答案之預測集所計算出來的分數

訓練時使用的一些東東

A. Optimizer ： Adamw (lr = 2e-5 ~ 1e-6，比較好的 lr 為 1e-5、15e-6)

B. Warm_up ：

1. get_linear_schedule_with_warmup (效果比較好，沒使用 correct_bias)
2. get_polynomial_decay_schedule_with_warmup (效果沒有很好)

C. 防止梯度爆炸 ： clip_grad_norm_ (max_norm = 1)

D. Batch_size = 16

E. 預測方式：

1. 直接預測
2. 選擇 Top k 去做比對

```
for i in range(EPOCHS):
    print(f"Epoch {i + 1} / {EPOCHS}:")
    loss, pred = train(model, train_loader, optimizer, scheduler)
    exact_match, all_match, real_f1, f1, count= 0, 0, 0, 0, 0
    embeddings = train_dataset.encodings['offset_mapping']

    for j, (ans, pre_ans) in enumerate(pred):
        if ans[0] == 0 and ans[1] == 0 and pre_ans[0] == 0 and pre_ans[1] == 0:
            all_match += 1
        elif ans[0] != 0 and ans[1] != 0:
            pre_1 = embeddings[j][pre_ans[0]][0]
            pre_2 = embeddings[j][pre_ans[1]][-1]
            true_ans = train_dataset.dict['ans'][j]
            pred_ans = train_dataset.dict['snippet'][j][pre_1:pre_2]

            if true_ans == pred_ans:
                exact_match += 1
                all_match += 1

            f1 += compute_f1(true_ans, pred_ans)
            count += 1

    f1 = format(f1 / count, ".4f")
    print("Train Loss: {}, F1: {}, EM: {:.4f}, All: {:.4f}".
            format(loss, f1, exact_match / count * 1.0, all_match / len(pred) * 1.0))
```

IV. Evaluation score

在模型建立完成後，對模型做了幾種微調，最後從這些調整中，選出表現最好的模型作為預測結果之模型，最後選擇 lr=1e-5 warmup top3 的模型做預測，微調主要分為兩個面向：

A. Learning rate 的調整

1. 單純調整 learning rate 大小
2. 調整 warm up 的幅度

B. 選擇預測結果的方式

1. 單純選擇值最高的作為答案 index
2. 選擇前 k 高的值做比對，輸出最有可能的答案，由於最後預測時一定會有解答，使用此方法避免預測無答案之輸出

	f1-score	exact mach	all match
lr=1e-5 warmup	0.63765	0.59239	0.68938
lr=1e-5 warmup top2	0.65061	0.60449	0.68764
lr=1e-5 warmup top3	0.65118	0.60499	0.68764
lr=15e-6 warmup	0.63439	0.58857	0.69046
lr=15e-6 warmup top2	0.65064	0.604	0.69022
lr=15e-6 warmup top3	0.6512	0.60441	0.68995
lr=15e-6->1e-7 warmup	0.63038	0.58658	0.69335
lr=15e-6->1e-7 warmup top2	0.6478	0.6025	0.69049
lr=15e-6->1e-7 warmup top3	0.64817	0.60259	0.69038