Applied Deep Learning Homework 2 report

M10915045 施信宏

Q1: Data processing

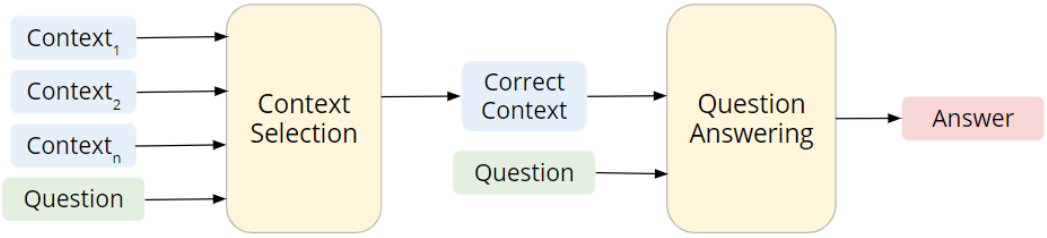
Q1-1_Tokenizer :

BertTokenizer分為兩個步驟，一為BasicTokenizer，一為WordpieceTokenizer，前者可以認定為標準化處理字串，比如unicode轉換、標點符號切割、去除非法字元…等等。後者則是將標準化後字串分割成一個一個字元，而注意的是英文中有合成詞的特性，將會被轉換成##的形式；中文將不考慮詞，而是直接切成單字形式，如“兩河文明”將會被處理為“兩”、“河”、“文”、“明”。數字的部分，若是數字過長，則同樣以##的方式標示與前字元結合，ex : 30000元，可能會拆解成300、##00、”元”。

Q1-2_Answer span :

原先的作法是將答案先用BertTokenizer，得到wordpiece的編碼，再用答案的tokens去找尋原先文本最接近答案位置，若找到全部符合的部分即將此視為答案，如長度超過最大長度，則截斷部分文章，ex： 長度650，最大長度512，答案位置位於400，則將前面的文章給捨棄。但這作法訓練出的model EM的準確率在context輸入皆為正確情況下只有0.745，請教過助教後，建議我參考 transformer sample code的方法。處理方法是使用Tokenizer內建的function，offsetMapping，這個function能找到原先文章內容經過 tokenize 後的位置，這時需要使用訓練集當中的start，找尋到文章答案的對應位置。且文章若過長，也不能直接丟棄，需要保留起來訓練，使model能夠關注正確的地方。在測試和驗證時，如要得到正確的答案，則是將通過model後的start_logit及end_logit記錄起來，並且建立一個hyper-parameter (n_best_size)，用來記錄前幾筆分數較高的位置，避免最高分數的位置無法還原，可以用順位的位置作為代替。使用此方法進行重建，在驗證集使用public.json的情況下，使用bert-base-chinese EM準確率為0.82。

Q2: Modeling with BERTs and their variants



A :

架構和助教提供的pipeline相同，pretrained model 用 bert-base-chinese。

先使用bertformultiplechoice建立context selection model，學習如何挑出正確的context，選項設定為4個，一個正解搭配三個錯誤，若paragraphs能挑選的錯誤答案小於3篇，則將缺額的選項直接padding為0，optimizer為adamW，batch_size = 3 ，learning rate = 1e-5，loss 則為 bertformultiplechoice 中 output 輸出的結果做 backward ，經測試後1個epoch準確率較高，約為0.934。

接著使用bertforquestionanswering 建立QA model，optimizer為adamW，epoch = 3 ，batch_size = 5 ，learning rate = 3e-5，loss 則為bertforquestionanswering 中 output 輸出的結果做 backward ，在epoch = 3，準確率達到訓練最高。

最後將context selection model 的結果套入 QA model ，EM準確率為0.78，F1為0.83左右。

B:

pretrained model 改為 hfl/chinese-roberta-wwm-ext。

RoBERTa在是基於BERT得到的更加強大的模型，使用Dynamic masking，使其在訓練期間使[MASK]的標記發生變化，同時也優化BERT的部分參數，且使用更多的資料來做訓練(BookCorpus, CC-News, OpenWebText…)，獲取一個比BERT還強大的model。

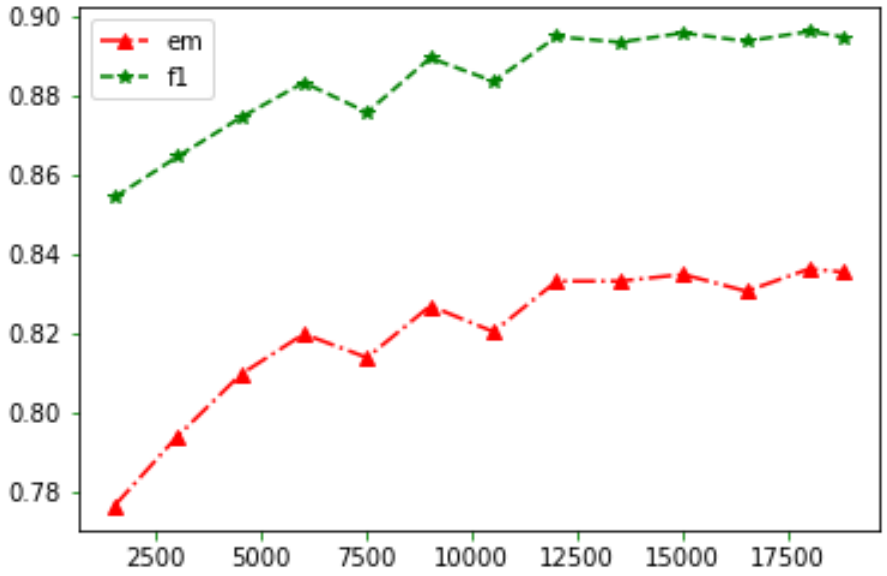
此model對中文更加優化，Whole Word Masking (wwm)可以將Wordpiece切成多份的詞彙，同時做[MASK]，且可以關注中文分詞(CWS)，與一般處理”字切分”不同，能夠保留一整個詞。

更換為chinese-roberta-wwm-ext後，參數皆不變的情況下，context selection model提升為0.935，QA model提升至 0.836，結合在一起後 EM準確率為0.803、F1 = 0.864，通過strong baseline。

Q3: Curves

下圖的驗證集為公開的測試集(public.json)，針對QA model的訓練過程做紀錄，而非經由context selection model得到的結果做輸入，故EM、F1數值會較高。

圖中的點為每經過1500個steps，便紀錄下的EM、F1。訓練3個epochs，總訓練steps有18807個，起始點為訓練至第1500個step時的數值。根據實驗結果，大約於第18000個step左右，分數會達到最高，完成3個epochs的訓練反而會稍稍降低一些準確度，故QA model (BERT 、Roberta)都是取第18000步的結果。



Q4: Pretrained vs Not Pretrained

若要從頭訓練一個transformer相關model，只需讀取其config，將AutoModelForQuestionAnswering.from_pretrained(….)階段改成AutoModelForQuestionAnswering.from_config(config) 即可訓練一個亂數生成的新權重模型。

hfl/chinese-roberta-wwm-ext 的 config 如下:


```
"architectures": [
  "BertForMaskedLM"
],
"attention_probs_dropout_prob": 0.1,
"bos_token_id": 0,
"directionality": "bidi",
"eos_token_id": 2,
"gradient_checkpointing": false,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 768,
"initializer_range": 0.02,
"intermediate_size": 3072,
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 12,
"num_hidden_layers": 12,
"output_past": true,
"pad_token_id": 1,
"pooler_fc_size": 768,
"pooler_num_attention_heads": 12,
"pooler_num_fc_layers": 3,
"pooler_size_per_head": 128,
"pooler_type": "first_token_transform",
"position_embedding_type": "absolute",
"transformers_version": "4.5.1",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 21128
```


此model和tokenizer使用hfl/chinese-roberta-wwm-ext的預先處理的config及tokenizer，在經過3個epochs，learning rate = 3e-5 ，batch_size = 5，訓練後，在pubic.json上的表現如下圖。


```
{'count': 3526, 'em': 0.06324446965399887, 'f1': 0.10703485084691684}
```

準確率非常的低，比起使用hfl/chinese-roberta-wwm-ext，fintune其參數後的準確率(EM = 0.836，F1 = 0.89)，差距十分巨大。
但結果也可想而知，原先的pretrained_model經過大量的資料及運算資源，訓練非常久的時間才獲得的參數，光用三個epochs本就很難達到其效果，且重複運算相同模型也須消耗許多實體資源和時間。故在 huggingface-transfomers 才會有很多人提供其訓練已久的模型放在網路上供大家使用，一起為社群努力，減少資源的浪費。

Q5: Bonus: HW1 with BERTs

Intent classification:

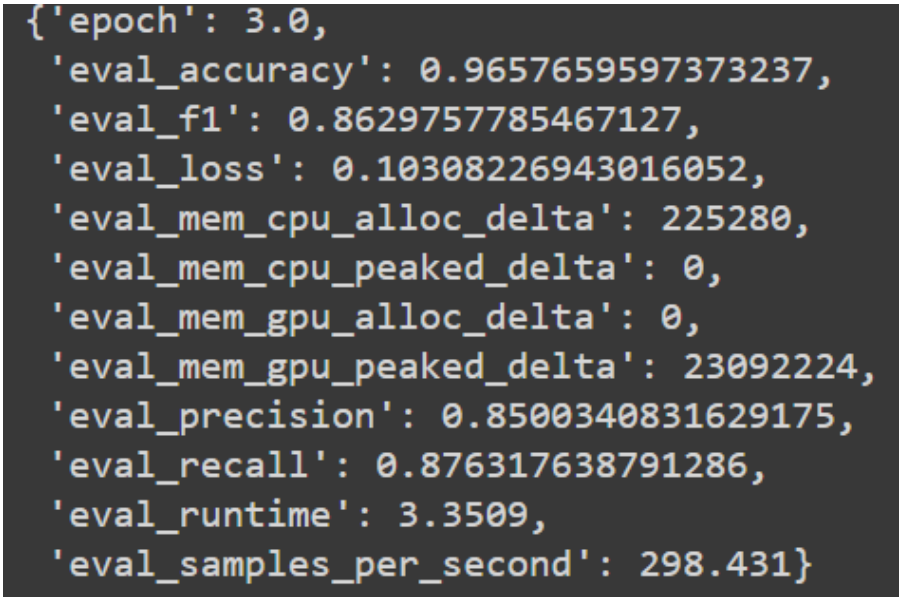
將作業一的train.json、eval.json，用來測試bert的句子分類功能(AutoModelForSequenceClassification)。Learning rate = 3e-5 ，batch_size = 10，Max_seq_length = 128，pretrained model = bert-base-uncased 隨著訓練epoch提升，optimizer = adamW 。準確率如下:

```
 Epoch: 1/5 | Batch: 1500/1500 | loss = 1.64131 | acc = 0.50  |0:02:08
Train | Loss:2.91045 Acc: 42.873
 Epoch: 1/5 | Validating...Epoch: 1/5 | dev_loss=1.27524 | dev_acc=72.267 |0:02:15
 Epoch: 2/5 | Batch: 1500/1500 | loss = 0.37158 | acc = 1.00  |0:04:25
Train | Loss:0.81358 Acc: 82.267
 Epoch: 2/5 | Validating...Epoch: 2/5 | dev_loss=0.62592 | dev_acc=84.833 |0:04:32
 Epoch: 3/5 | Batch: 1500/1500 | loss = 0.23129 | acc = 1.00  |0:06:42
Train | Loss:0.34911 Acc: 92.300
 Epoch: 3/5 | Validating...Epoch: 3/5 | dev_loss=0.46036 | dev_acc=88.700 |0:06:48
 Epoch: 4/5 | Batch: 1500/1500 | loss = 0.56084 | acc = 0.80  |0:08:59
Train | Loss:0.18685 Acc: 95.753
 Epoch: 4/5 | Validating...Epoch: 4/5 | dev_loss=0.40193 | dev_acc=90.867 |0:09:06
 Epoch: 5/5 | Batch: 1500/1500 | loss = 0.01532 | acc = 1.00  |0:11:16
Train | Loss:0.12615 Acc: 97.333
 Epoch: 5/5 | Validating...Epoch: 5/5 | dev_loss=0.39788 | dev_acc=91.100 |0:11:23
```

在作業一使用RNN相關model時，驗證集的準確率很難達到85%以上，即使訓練集已經達到99%的準確，而改使用bert之後有了明顯的提升，雖未將測試集的結果上傳至kaggle，但猜想也應有0.92以上的分數。

Slot tagging:

此任務相比intent classification較為複雜一些，需考慮tokens經tokenizer後編碼不同的問題，採用transfomer 中 token classification的方法，使用tokenizer內建的參數(is_split_into_words=True)放入分開的tokens而不是一整個句子，返回輸出後，還需考慮[CLS]，[SEP]及單字被切割成多個單詞的問題，所幸tokenizer功能十分強大，可以使用word_ids()去追蹤特別符號或是單字拆解成多個的問題，再使用正確的labels去標記位置即可進行訓練。Batch_size = 16 ， learning rate = 2e-5，epochs = 3，optimizer = adamW，使用驗證集在seqeval回傳的評量分數如下:



表現算是出色，在colab Telsa P100只花了7分鐘進行了3個epochs的訓練，效果比起作業一使用LSTM來的優秀，經過加分題知曉了transfomer model的強大，並且對四個下游任務(QA，mutiplechoice , seqclassification , tokenclassification)都有一定程度的了解，收穫了許多。