Bert-based models for NLP tasks

目錄

Q1: Data processing (2%)

1. Tokenizer (1%)

我使用 Huggingface 提供的 Bert tokenizer 作為我本次作業的 tokenizer：Bert tokenizer 的字典大小為 21128，其中除了中文繁體字外，有簡體中文字、標點符號、特殊符號、數字、5 個特殊 tokens 如表 1 所示等，Bert tokenizer 參考 Google NMT 提出的 WordPiece Tokenization，在中文文字上將每個獨立的字及逼點符號轉成個別的 token，類似 character-level tokenization，例如：[他移開]經過 tokenize 後為[800, 4919, 7274]。Bert tokenizer 的字典有 9616 個 token 的前面有## 的符號即為 wordpieces，較多出現於中文字、英文的字根、數字或特殊符號。

表 1、Bert tokenizer 的特殊 tokens

Token	**功能**
[CLS]	放在第一個句子的首位，經過 Bert 模型後，此位置得到的向量可以用於後續的分類任務。
[SEP]	有兩個句子的文本會被串接成一個輸入序列，並在兩句之間插入這個 token 作為分割。
[UNK]	沒出現在 Bert 字典裡頭的字會被這個 token 取代。
[PAD]	zero padding 時使用，將長度不同的序列補齊做 batch 運算。
[MASK]	未知 token，僅在預訓練階段用到。

Context Selection 任務中，一次會將 question 及四個潛在的 context 放進 Bert tokenizer，經過 Bert tokenizer 後會出現三個向量分別是：'input_ids'、'token_type_ids'及'attention_mask'。input_ids 為輸入的每個文字轉換成字典中的 ID；token_type_ids 中只會出現 0 或 1，在此任務中用於分別 question、context 及額外 padding 的 token；attention_mask 也只會出現 0 或 1，1 代表原始句子的 token，0 代表額外 padding 的 token。我將 Bert tokenizer 的 max_length 設定為 512(模型所能接受的最長長度)，並 padding 至 512 長，若 question 加 context 的長度大於 512 個 token，會優先從 context 的最後部分開始 truncate。最後一筆資料的向量維度為[4, 512]。

Question Answering 任務中，一次會將 question 及答案出現的 context 放進 Bert tokenizer，經過 Bert tokenizer 後會出現'input_ids'、'token_type_ids'、'attention_mask'、'offset_mapping'及'overflow_to_sample_mapping'：'input_ids'、'token_type_ids'、'attention_mask'內容和 Context Selection 任務的相同，truncate

的方法也是從 context 的最後部分開始。'offset_mapping'的內容為每個 token 的在 tokenize 前在句子中的起始與結束位置，會需要此功能是因為 context 經過 tokenize 後答案的位置可能不會跟 tokenize 前的一樣，所以利用此回傳的內容找尋正確答案在 context 經過 tokenize 後的位置；'overflow_to_sample_mapping'為返回每筆 tokenize 後的資料在 tokenize 前的序位，因為當 question 加 context 的長度大於 512 個 token 時，有部分同個 context 的 token 會移至下一筆資料，若兩筆資料原本同屬於一個 context，回傳的 overflow_to_sample_mapping 值會相同，利用此回傳的內容可確認每筆 tokenize 後的資料原本是否為同個 context。'max_length'我設為 512；' stride' 設為 128，stride 的功能為在 context 被 truncate 的位置前加入幾個 token 一起放入下一筆被 truncate 的資料中，目的為提供重複的 token 讓被 truncate 的資料比較完整。

2. Answer Span (1%)

a. Convert the answer span start/end position on characters to position on tokens after BERT tokenization

我利用 Bert tokenizer 提供的'offset_mapping'內容重新標記正確答案的起始及結束位置：'offset_mapping'會返回每個 token 在 tokenize 前位於 context 中的起始及結束位置。資料的 answer start 值為位於 context 在 tokenize 前 character 的位置，answer end 值為 answer start 值再加上 answer text 的長度，當資料的 answer start/end 值對上'offset_mapping'中 token 的原始位置時，此 token 的位置即為 context 在 tokenize 後 answer 的起始及結束位置。

有些 context 超過預設長度會被 truncate，被 truncate 的 context 內容會變為下筆資料，若 character 的 answer start/end 值不在當筆資料的'offset_mapping'中，會將此筆資料的 answer start/end 值改為 0([cls]的 index)。

b. Rules I apply to determine the final start/end position

Bert model 給予的 outputs 分為 start_logits 及 end_logits：start_logits 為當筆資料中每個 token 擔任答案起始位置的分數，end_logits 為每個 token 擔任結束位置的分數，我最終會選擇 start_logits 中分數最高的 token 作為 final start position；end_logits 中分數最高的 token 作為 final end position。若一個 context 被切為兩部分，則我將兩部分所有 token 的分數共同考慮，並選擇最高的 token 作為 final start/end position。

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

1. Describe (2%)

a. My model (configuration of the transformer model)

bert-base-chinese:

- 這個模型使用繁體與簡體的維基百科資料作填字任務的預訓練。
- 總共有 12 層 Transformer hidden layers，共 1 億多個參數如圖 1。

BERT base chinese				
Module	Key	Shape	Count	Parameters
Embedding	embeddings.word_embeddings.weight	[21128, 768]	16,226,304	16,622,592
	embeddings.position_embeddings.weight	[512, 768]	393,216	
	embeddings.token_type_embeddings.weight	[2, 768]	1,536	
	embeddings.LayerNorm.weight	[768,]	768	
	embeddings.LayerNorm.bias	[768,]	768	
Transformer*12	encoder.layer.0.attention.self.query.weight	[768, 768]	589,824	7,087,872 * 12
	encoder.layer.0.attention.self.query.bias	[768,]	768	
	encoder.layer.0.attention.self.key.weight	[768, 768]	589,824	
	encoder.layer.0.attention.self.key.bias	[768,]	768	
	encoder.layer.0.attention.self.value.weight	[768, 768]	589,824	
	encoder.layer.0.attention.self.value.bias	[768,]	768	
	encoder.layer.0.attention.output.dense.weight	[768, 768]	589,824	
	encoder.layer.0.attention.output.dense.bias	[768,]	768	
	encoder.layer.0.attention.output.LayerNorm.weight	[768,]	768	
	encoder.layer.0.attention.output.LayerNorm.bias	[768,]	768	
	encoder.layer.0.intermediate.dense.weight	[3072, 768]	2,359,296	
	encoder.layer.0.intermediate.dense.bias	[3072,]	3072	
	encoder.layer.0.output.dense.weight	[768, 3072]	2,359,296	
	encoder.layer.0.output.dense.bias	[768,]	768	
	encoder.layer.0.output.LayerNorm.weight	[768,]	768	
	encoder.layer.0.output.LayerNorm.bias	[768,]	768	
Pooler	pooler.dense.weight	[768, 768]	589,824	590,592
	pooler.dense.bias	[768,]	768	
				total: 102,267,648

圖 1、bert-base-chinese 架構(資料來源: https://zhuanlan.zhihu.com/p/550359168)

b. Performance of my model.

Context Selection(訓練 4 個 epoch)

- 最佳 Training Accuracy: 0.9866(出現於第 4 個 epoch)
- 最佳 Validation Accuracy: 0.9535(出現於第 1 個 epoch)

Question Answering(訓練 30 個 epoch)

- 最佳 Training Accuracy: 0.9961(出現於第 28 個 epoch)
- 最佳 Validation Accuracy: 0.8053(出現於第 7 個 epoch)

c. The loss function I used.

Context Selection 及 Question Answering 我皆使用 CrossEntropyLoss()作為 loss function：

CrossEntropyLoss()函數(如公式(1))可用來量化兩個概率分佈之間差異的損失函數，多用於分類問題，pytorch 的 CrossEntropyLoss 結合了 LogSoftmax 和

NLLLoss。

$$l_n = -\sum_{c=1}^{C} w_c \log \frac{\exp(x_{n,c})}{\sum_{i=1}^{C} \exp(x_{n,i})} y_{n,c} \qquad \text{公式(1)}$$

說明：x 為預測值；y 為實際值；w 為權重；C 為分類數量；n 為第幾個 batch

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

Context Selection:

- ✓ Optimization algorithm: Adam。
- ✓ Learning rate: $1*10^{-5}$。
- ✓ Effective Batch size: 1(batch size: 1, gradient accumulation steps: 1)。

Question Answering:

- ✓ Optimization algorithm: AdamW。
- ✓ Learning rate: $5*10^{-5}$。
- ✓ Effective Batch size: 64(batch size: 32, gradient accumulation steps: 2)。

2. Try another type of pretrained model and describe (2%)

a. My model (configuration of the transformer model) and the difference between pretrained model (architecture, pretraining loss, etc.)

hfl/chinese-roberta-wwm-ext:

此模型架構和 bert base 相同，但做了五點改善(模型比較如表 2)：

I. wwm 即為 Whole Word Masking，相比於 bert 的改進是用 Mask 標籤替換一個完整的詞而不是單一字詞。

II. 預訓練使用的數據集增至 5.4B 詞數。

III. 取消 Next Sentence Prediction（NSP）

IV. 不採用先使用 max_len=128 的數據預訓練，然後用 max_len=512 的數據預訓練的模式，直接使用 max_len=512 的數據進行預訓練。

V. 訓練步數延長至 1M 步

hfl/chinese-roberta-wwm-ext-large:

此模型架構和 bert large 相同，bert large 有 24 層 layers、1024 個 hidden dimension、16 個 attention heads、總共 3.36 億個 parameters。

和 hfl/chinese-roberta-wwm-ext 比較起來除了用了較大的模型架構，還有延長訓練步數至 2M 步即增大 batch size 至 512。(模型比較如表 2)

表 2、Bert 模型訓練設置比較

	BERT	BERT-wwm	BERT-wwm-ext	RoBERTa-wwm-ext	RoBERTa-wwm-large-ext
Masking	WordPiece	WWM	WWM	WWM	WWM
Type	BERT-base	BERT-base	BERT-base	BERT-base	BERT-large
Data Soruce	wiki	wiki	Wiki+ext	wiki+ext	wiki+ext
Training Tokens	0.4B	0.4B	5.4B	5.4B	5.4B
Training Device	TPU Pod v2	TPU v3	TPU v3	TPU v3	TPU Pod v3-32
Training Steps	1M	100K+100K	1M+400K	1M	2M
Max Length	512	128 / 512	128 / 512	512	512
Batch Size	256	2560 / 384	2560 / 384	384	512
Optimizer	AdamW	LAMB	LAMB	AdamW	AdamW
Vocabulary	21,128	BERT vocab	BERT vocab	BERT vocab	BERT vocab
Initial Checkpoint	Random initialization	BERT weight	BERT weight	BERT weight	Random initialization

參考資料:
https://mp.weixin.qq.com/s?__biz=MzU2NDQ3MTQ0MA==&mid=2247484927&idx=1&sn=b396882679c49c7d1a377a1b92f14c17&chksm=fc4b3600cb3cbf164a917aa17b444f1106ce0cca384ca4813b785aaecec19f2e70da287081a4&scene=21#wechat_redirect

b. Performance and Settings of my model.

hfl/chinese-roberta-wwm-ext:
Context Selection(訓練 5 個 epoch)

- ✓ Optimization algorithm: Adam。
- ✓ Learning rate: $1*10^{-5}$。
- ✓ Effective Batch size: 4(batch size: 1, gradient accumulation steps: 4)。
- ➢ 最佳 Training Accuracy: 0.9964(出現於第 5 個 epoch)

- 最佳 Validation Accuracy: 0.9631(出現於第 2 個 epoch)

Question Answering(訓練 20 個 epoch)

- ✓ Optimization algorithm: AdamW。
- ✓ Learning rate: $1*10^{-5}$。
- ✓ Effective Batch size: 4(batch size: 2, gradient accumulation steps: 2)。
- 最佳 Training Accuracy: 0.9962(出現於第 18 個 epoch)
- 最佳 Validation Accuracy: 0.8255(出現於第 18 個 epoch)

hfl/chinese-roberta-wwm-ext-large:
Context Selection(訓練 5 個 epoch)

- ✓ Optimization algorithm: AdamW。
- ✓ Learning rate: $3*10^{-5}$。
- ✓ Effective Batch size: 64(batch size: 2, gradient accumulation steps: 32)。
- 最佳 Training Accuracy: 0.9977(出現於第 5 個 epoch)
- 最佳 Validation Accuracy: 0.9641(出現於第 4 個 epoch)

Question Answering(訓練 11 個 epoch)

- ✓ Optimization algorithm: AdamW。
- ✓ Learning rate: $5*10^{-5}$。
- ✓ Effective Batch size: 96(batch size: 1, gradient accumulation steps: 96)。
- 最佳 Training Accuracy: 0.9960(出現於第 11 個 epoch)
- 最佳 Validation Accuracy: 0.8352(出現於第 9 個 epoch)

在 Kaggle 上選取的最佳訓練成果所使用的模型:
Context Selection: **hfl/chinese-roberta-wwm-ext +**
Question Answering: **hfl/chinese-roberta-wwm-ext-large**

Other findings:

- ✧ **Learning rate** 調至 $1*10^{-4}$ 時，模型 fine tune 訓練結果會很差，因此我最高只調到 $5*10^{-5}$。
- ✧ **Effective Batch size** 一開始我參考助教的範例及 Hugging Face 上的都調在個位數之內，後來看到同學調到 64 訓練出來的結果上傳至 kaggle 可得到 accuracy 0.81，因此我訓練 hfl/chinese-roberta-wwm-ext-large 時嘗試將 Effective Batch size 往上調，有得到較好的成果。

Q3: Curves (1%)

1. Plot the learning curve of your QA model

我使用 **hfl/chinese-roberta-wwm-ext-large** 的訓練結果畫以下的圖。

a. Learning curve of loss (0.5%)

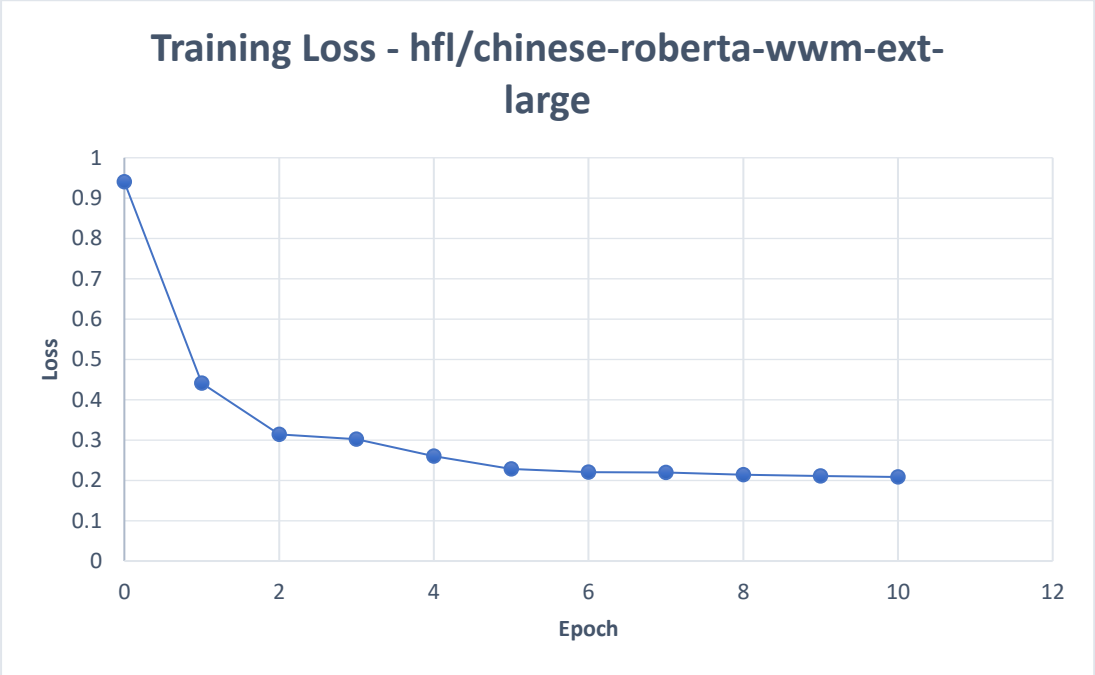

圖 2、Learning curve of loss

表 3、Learning curve of loss 數值

epoch	0	1	2	3	4	5	6	7	8	9	10
Training Loss	0.94	0.44	0.31	0.30	0.26	0.23	0.22	0.22	0.21	0.21	0.21

b. Learning curve of EM (0.5%)

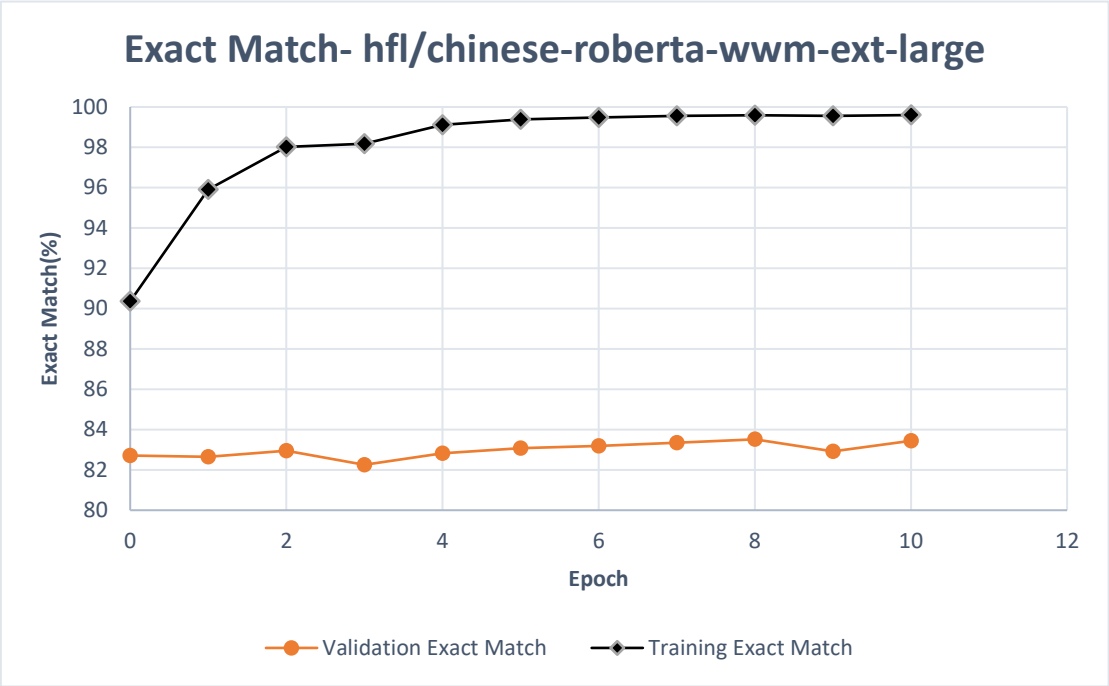

圖 3、Learning curve of EM

表 4、Learning curve of EM 數值

epoch	0	1	2	3	4	5	6	7	8	9	10
Training Exact Match(%)	90.36	95.90	98.02	98.17	99.11	99.38	99.45	99.55	99.59	99.56	99.60
Validation Exact Match(%)	82.72	82.65	82.95	82.25	82.82	83.08	83.18	83.35	83.52	82.92	83.45

Q4: Pretrained vs Not Pretrained (2%)

Describe:

✧ The configuration of the model and how do you train this model

我使用 bert-base-chinese 模型作 Pretrained vs Not Pretrained 的比較，並在 QA 的 dataset 分別作 without pretrained weights 及 with pretrained weights 的模型訓練，訓練設置如下：

✓ Epoch: 30

✓ Optimization algorithm: AdamW。

✓ Learning rate: $5*10^{-5}$。

✓ Effective Batch size: 64(batch size: 8, gradient accumulation steps: 8)。

✧ The performance of this model vs. BERT

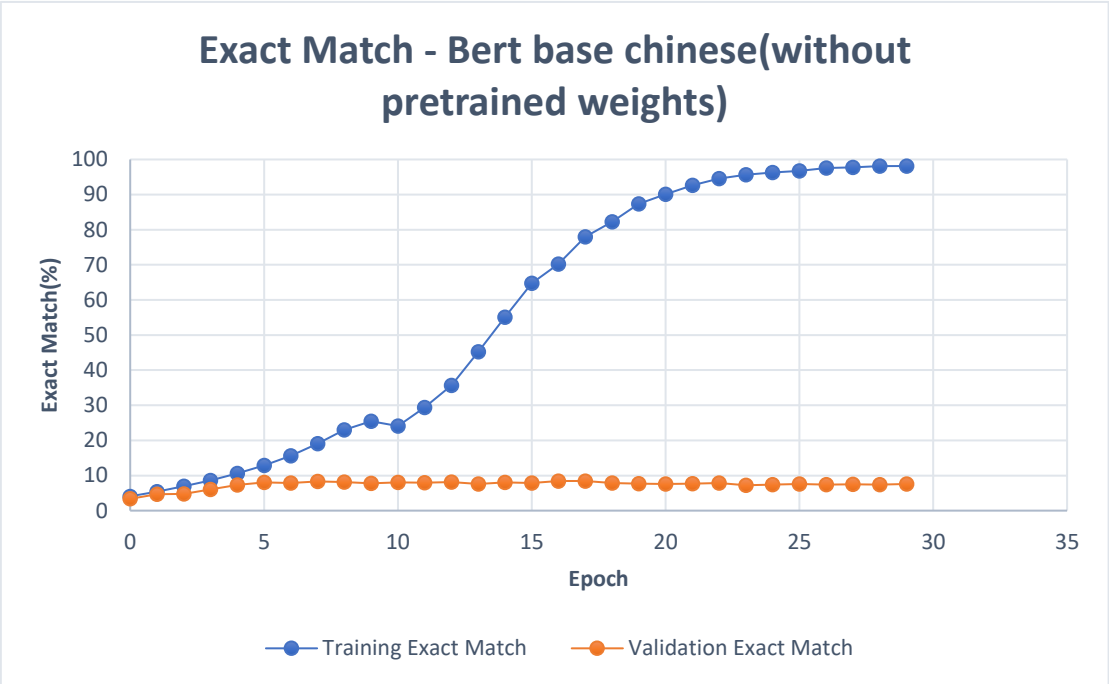

圖 4、bert-base-chinese without pretrained weights 訓練結果

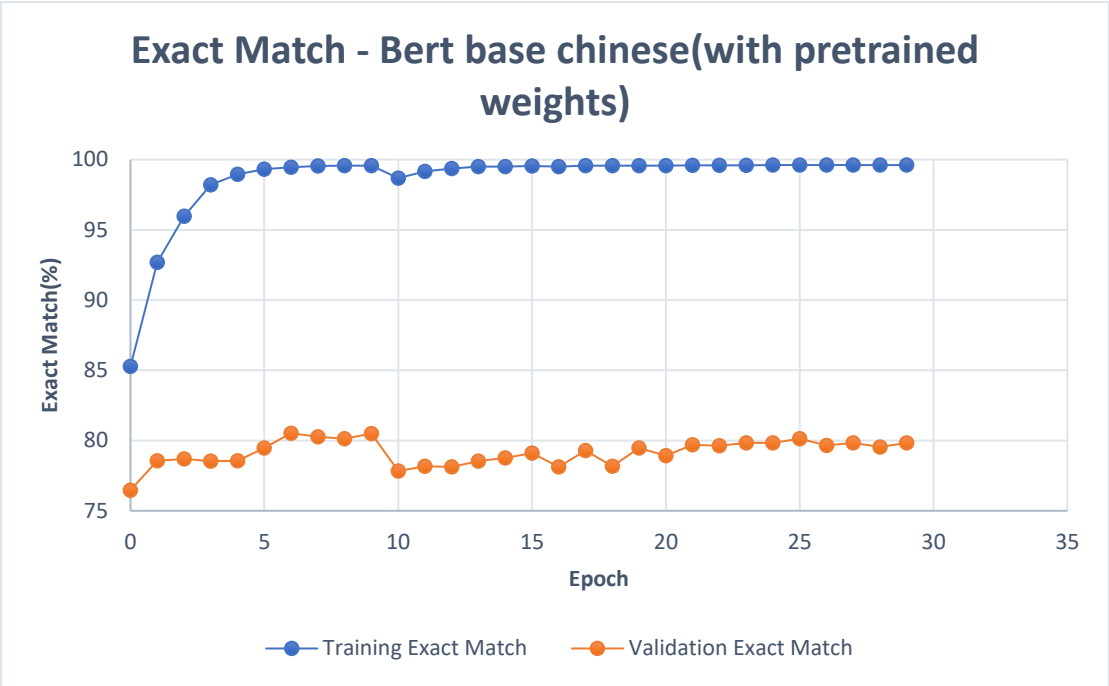

圖 5、bert-base-chinese with pretrained weights 訓練結果

由圖 4、圖 5 可發現兩個 model 在經過 30 個 epoch 訓練後，Training Exact Match Accuracy 幾乎皆可達 100%，差別在於 bert-base-chinese with pretrained weights 在第 5 個 epoch 時就可以達到；但在 Validation Exact Match Accuracy 就有所差異：bert-base-chinese with pretrained weights 可以達到 80%左右，但 bert-base-chinese without pretrained weights 只能在 8%左右，差了 10 倍之多，有 without pretrained weights 的 bert model 很容易就 overfit 在 training data，導致其在遇見沒看過的資料時表現不佳，所以複雜的 bert model 有 pretrained 過其他額外資料後再執行其他任務的 fine tune 訓練，對於其 Validation 的表現是非常有幫助！

Q5: Intent Classification and Slot Tagging with

BERTs (2%)

Train a BERT-based model on HW1 dataset and describe

a. My model

我使用 **bert-base-uncased** 作為在 Intent Classification 和 Slot Tagging 資料訓練的模型：此模型架構和 Q2 提及的 **bert-base-chinese** 架構相同，pretrain 的資料使用英文句子，並執行 Masked language modeling (MLM)及 Next sentence prediction (NSP)兩種任務。Uncased 的意思是模型不會分辨 english 和 English 的差異。Hugging Face 的模型明文件中提到，即便句字是中性的，模型本身具有 biased predictions，我認為這跟當初預訓練的資料集有關。

b. Performance of your model.

I. Intent classification (1%)

訓練 Epoch 次數: 5

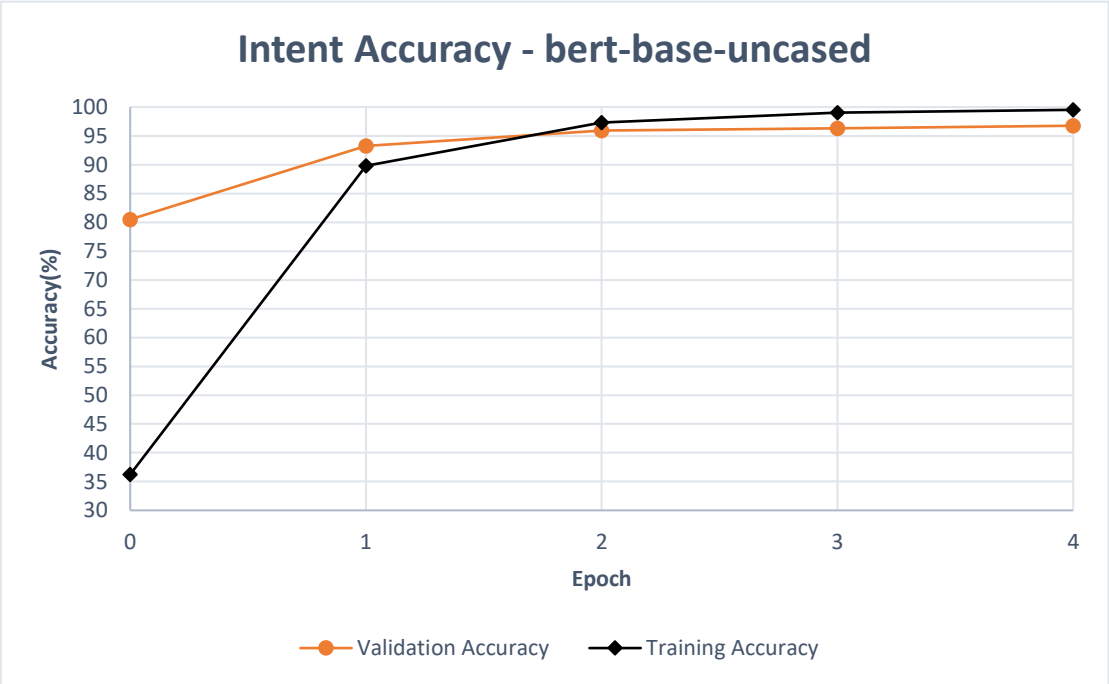

圖 6、Intent classification training and validation accuracy

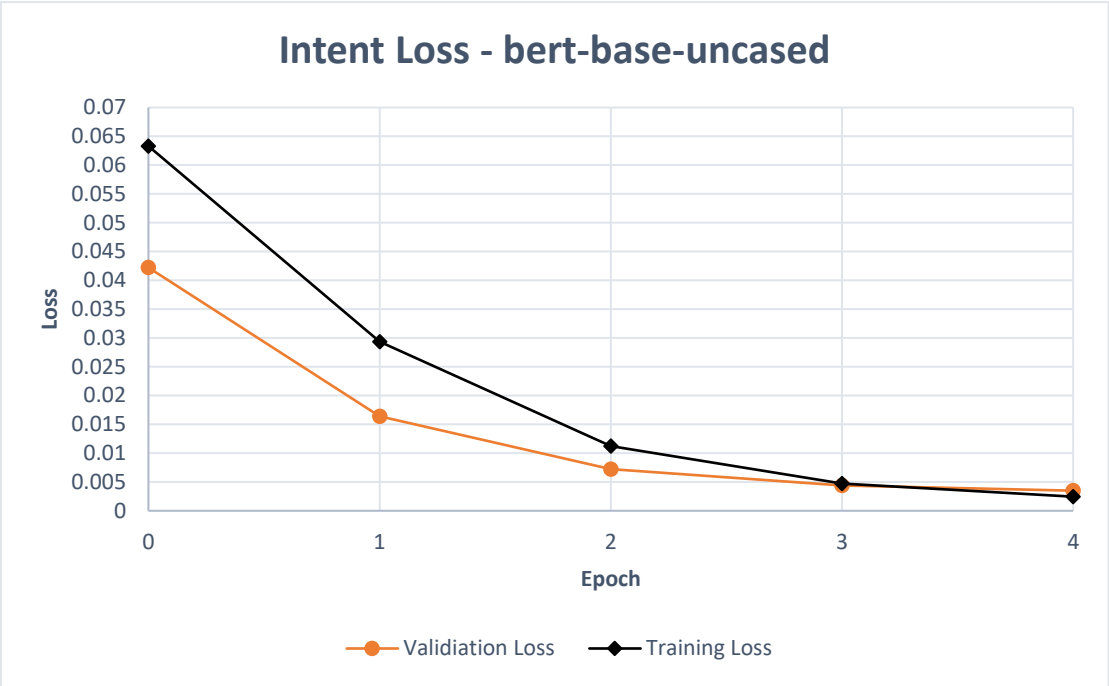

圖 7、Intent classification training and validation loss

由圖 6 可知，利用 bert 模型只要訓練 5 個 epoch 就在 intent 的任務中達到 95%以上的 accuracy，第一個 epoch 訓練時 training 的 accuracy 比 validation 低，可能是一開始接觸沒看過的資料，模型的參數才剛開始作調整，訓練兩個 epoch 後，training 的 accuracy 就比 validation 高了。訓練完第一個 epoch 後 validation 的 accuracy 就可以達到 80%以上，可見 bert 經過預訓練後很快就能收斂且達到很高的 accuracy！

II. Slot tagging (1%)

訓練 Epoch 次數: 5

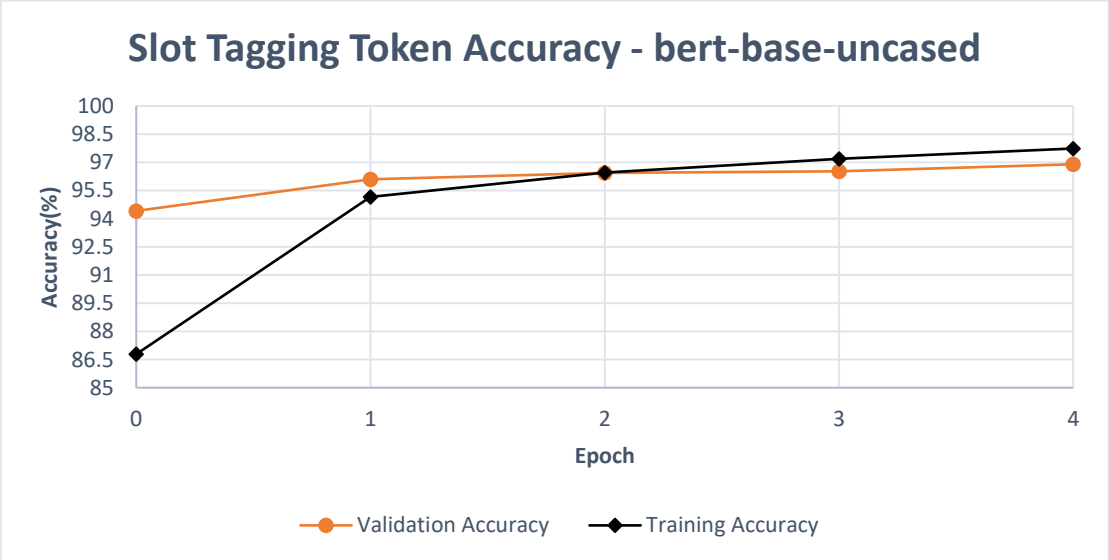

圖 8、Slot tagging training and validation Token accuracy

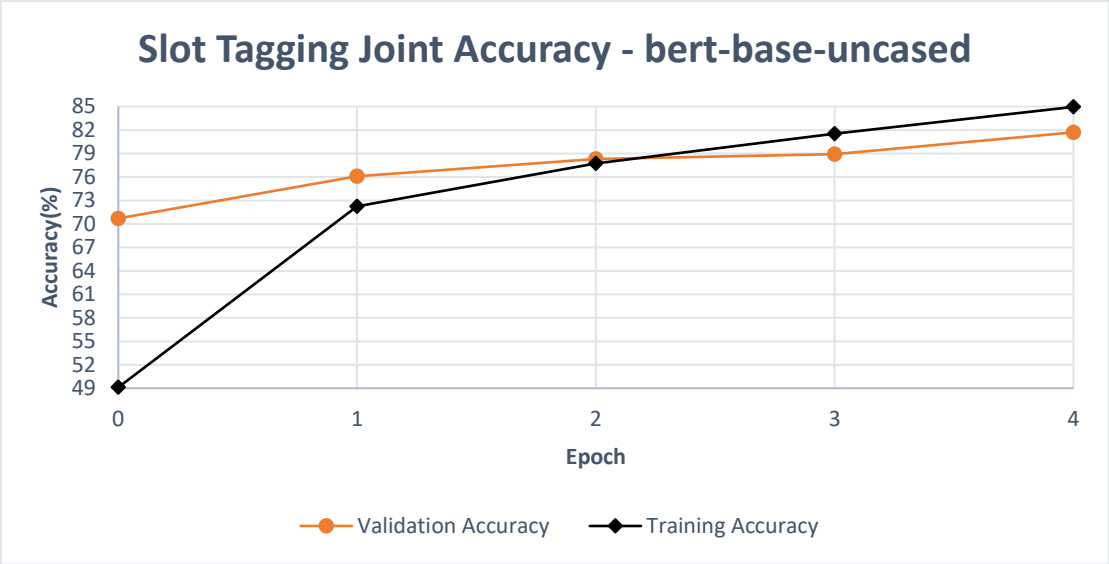

圖 9、Slot tagging training and validation Joint accuracy

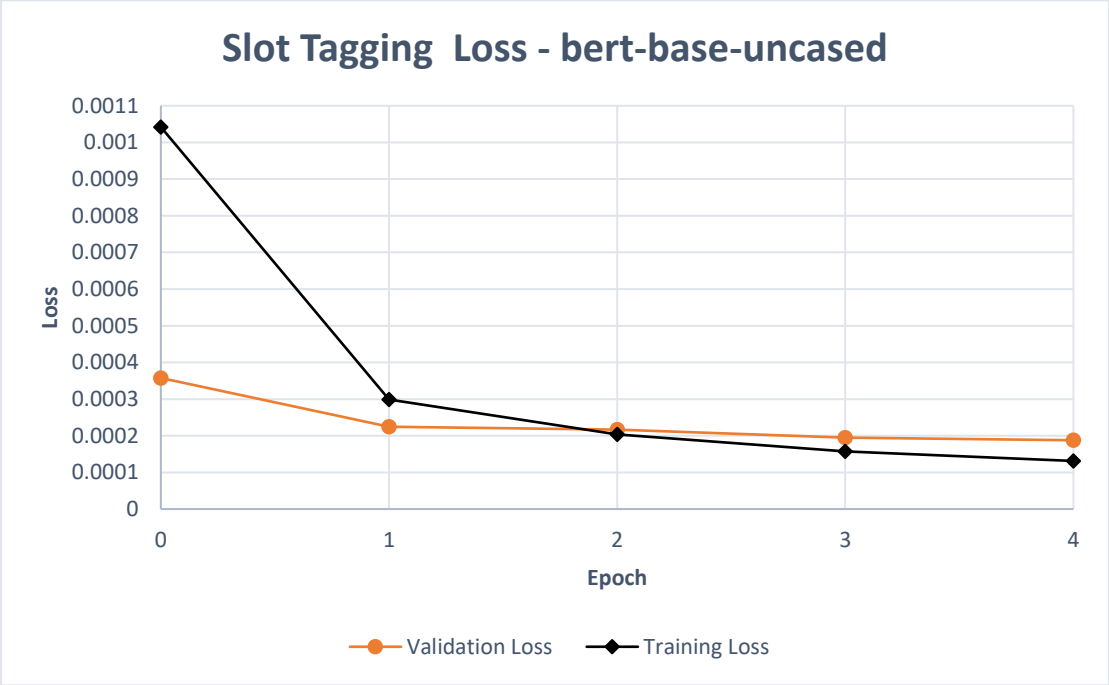

圖 10、Slot tagging training and validation loss

由圖 8 可知，利用 bert 模型只要訓練 5 個 epoch 就在 slot tagging 的任務中達到 97%左右的 token training accuracy，Joint training accuracy 可達 85%。第一個 epoch 訓練時 training 的 accuracy 比 validation 低，和 intent 的情況雷同，可能是一開始接觸沒看過的資料，模型的參數才剛開始作調整，訓練三個 epoch 後，training 的 accuracy 就比 validation 高了。訓練完第一個 epoch 後 validation 的 token accuracy 就可以達到 95%以上，joint accuracy 就可以達到 80%以上，可見 bert 經過預訓練後很快就能收斂且達到很高的 accuracy！

c. The loss function you used.

兩個任務皆為 CrossEntropyLoss()函數(如公式(1))。

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

兩個任務的設置皆為以下所示：

- Optimization algorithm: AdamW。
- Learning rate: $3*10^{-5}$。
- Effective Batch size: 64(batch size: 64, gradient accumulation steps: 1)。