ADL HW2 @NTU, 2021 spring

B06902135 資工四 蔡宜倫

1. Data processing (2%)

- **Tokenizer (1%):**
  - **Describe in detail about the tokenization algorithm you use. You need to explain what it does in your own ways.**
    - 我使用transformers的`BertTokenizerFast`做tokenization，並且使用的pre-trained model是`"bert-base-chinese"`：共有21228個token在vocab file。
- **Answer Span (1%):**
  - **How did you convert the answer span start/end position on characters to position on tokens after BERT tokenization?**

```python
answer_start_token = tokenized_paragraph.char_to_token(question["answers"][0]["start"])
answer_end_token = tokenized_paragraph.char_to_token(question["answers"][0]["start"] + len(question["answers"][0]["text"]) - 1)
```

    - 如上所示，我使用了`char_to_token`這個function轉換原本在character space的位置對應到token space的位置：這個function的input會是character在sequence中的index，輸出得到encoded token 的index。
  - **After your model predicts the probability of answer span start/end position, what rules did you apply to determine the final start/end position?**
    - 我使用了兩種方法在尋找start和end的span。
      - 第一種：找到start和end的機率，並且找最大可能的start prob. + end prob.的組合，能生成最大值的即是我的answer span。但這樣會生成一個問題，如果model不夠好，可能會造成end的index在start index的前面，因此我後來使用了第二種的方法，可以有效避免發生end prior to start的情況。
      - 第二種：先找到start的最大可能值，接著再做post-processing，從start index往後找一個區間內end index的機率最大值，這樣找到的answer span理論上較合理。

```python
start_probs, start_indexs = torch.topk(output.start_logits[k], k=1, dim=0)
for start_prob, start_index in zip(start_probs, start_indexs):
    length_prob, length = torch.max(output.end_logits[k][start_index : start_index + max_answer_len], dim=0)
    prob = start_prob + length_prob
```

2. Modeling with BERTs and their variants (4%)

- **Describe (2%)**
  - **your model (configuration of the transformer model)**
    - `BertForMultipleChoice`:
      - 使用transformer的pretrained-bert: `"hfl/chinese-macbert-large"`
      - **Configuration:**

```
"architectures": [
  "BertForMultipleChoice"
],
"attention_probs_dropout_prob": 0.1,
"directionality": "bidi",
"gradient_checkpointing": false,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 1024,
"initializer_range": 0.02,
"intermediate_size": 4096,
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 16,
"num_hidden_layers": 24,
"pad_token_id": 0,
"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": "4.5.0",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 21128
```

    - `BertForQuestionAnswering`:
      - 使用transformer的pretrained-bert: `"hfl/chinese-macbert-large"`
      - **Configuration:**

```
"architectures": [
  "BertForQuestionAnswering"
],
"attention_probs_dropout_prob": 0.1,
"directionality": "bidi",
"gradient_checkpointing": false,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 1024,
"initializer_range": 0.02,
"intermediate_size": 4096,
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 16,
"num_hidden_layers": 24,
"pad_token_id": 0,
"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": "4.5.0",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 21128
```

- **performance of your model.**
  - Context Selection accuracy on `public.json`: 95.8%
  - Question Answer EM on `public.json`: 84.3%
  - **Joint**:

    ```
    {"count": 3526, "em": 0.819058423142371, "f1": 0.8733224272153176}
    ```

- **the loss function you used.**
  - Context Selection: 對選取的context做`Cross-Entropy Loss`。
  - Question Answer: 對選取的answer start 和answer end做`Cross-Entropy Loss`，並總和相加成為最後的loss。
- **The optimization algorithm (e.g. Adam), learning rate and batch size.**
  - Context Selection:
    - Optimization Algorithm: `transformers.AdamW`
    - Scheduler: `transformers.optimization.get_linear_schedule_with_warmup`
    - Learning Rate: `3e-5`
    - Batch Size: `1`
    - Epoch: `2`

        - Gradient Accumulation Step: `40`
    - Question Answering:
        - Optimization Algorithm: `transformers.AdamW`
        - Scheduler: `transformers.optimization.get_linear_schedule_with_warmup`
        - Learning Rate: `3e-5`
        - Batch Size: `8`
        - Epoch: `2`
        - Gradient Accumulation Step: `2`

- **Try another type of pretrained model and describe (2%)**
    - **your model**
        - 對於CS和QA都使用transformer的pretrained-bert: `"bert-base-chinese"`
        - Configuration:

```
"architectures": [
  "BertForMaskedLM"
],
"attention_probs_dropout_prob": 0.1,
"directionality": "bidi",
"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,
"pad_token_id": 0,
"pooler_fc_size": 768,
"pooler_num_attention_heads": 12,
"pooler_num_fc_layers": 3,
"pooler_size_per_head": 128,
"pooler_type": "first_token_transform",
"type_vocab_size": 2,
"vocab_size": 21128
```

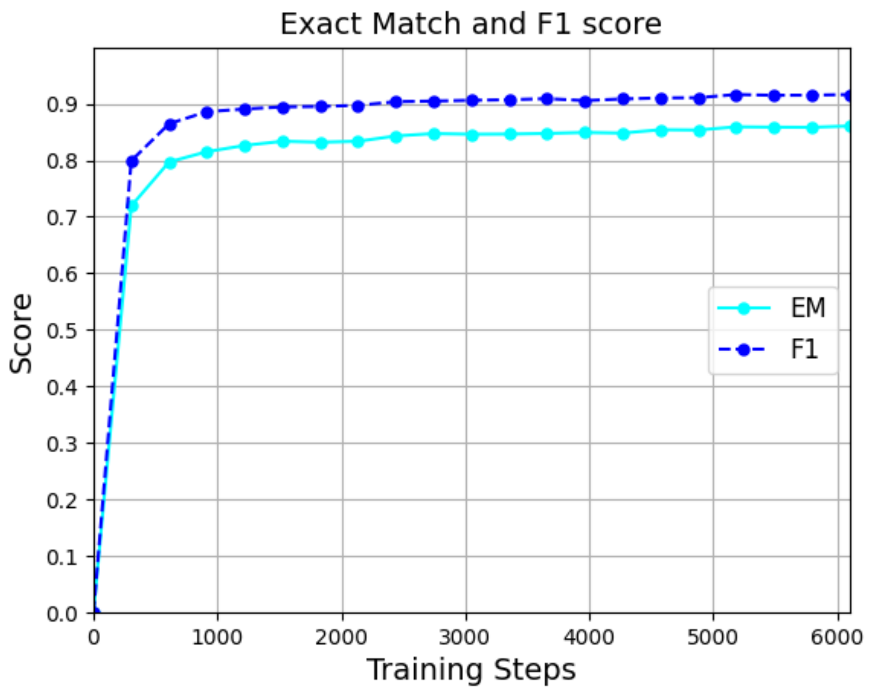
    - **performance of your model.**
        - Context Selection accuracy on `public.json`: 85.2%
        - Question Answer EM on `public.json`: 75.3%
        - **Joint**:

```
{"count": 3526, "em": 0.702468423967332, "f1": 0.7689004272513126}
```

- **the difference between pretrained model (architecture, pretraining loss, etc.)**
  - 可以從configuration很明顯得看出`chinese-macbert-large`相較於`bert-base-chinese`明顯是一個比較大的model，`intermediate_size`和`hidden_size`都有很大的差異。
  - 在masking的時候，相較於去遮蔽mask token的位置，`chinese-macbert-large`在訓練的時候是去mask 有相近meaning的詞。
  - 這樣的做法可以讓pre-train和fine-tune成為更相近的task，因此我們在使用fine-tune pre-trained模型在我們的task和data上時，可以將model原有的performance更好地保存下來。
  - 可以看到`chinese-macbert-large`在overall 的performance上有顯著的提升，em從70%上升到82%。

3. Curves (1%)

- **Plot**
  - **learning curve of EM (0.5%)** and **F1 (0.5%)**



  - learning curve是使用`"hfl/chinese-macbert-large"`和第二題的QA參數直接對`public.json`所得，並且只有做在QA task上，也就是直接給relevant的context了，不用做CS task。
    - Highest EM score: 86.0%
    - Highest F1 score: 91.6%

4. Pretrained vs Not Pretrained (2%)

- **Train a transformer model from scratch (without pretrained weights) on the dataset**

- **Describe**
  - **The configuration of the model and how do you train this model**
    - 總共需要兩種model的架構，一種是BertForMultipleChoice，另一種是BertForQuestionAnswering。
    - 我採用了同樣的model configuration去train兩個task：

```
config = BertConfig(
    vocab_size = 30522,
    hidden_size = 552,
    num_fidden_layers = 6,
    num_attention_heads = 6,
    intermediate_size = 1024
)
```

    - 其餘的hyper-parameters設定和之前一樣，如下所示：
      - Context Selection:
        - Optimization Algorithm: `transformers.AdamW`
        - Scheduler: `transformers.optimization.get_linear_schedule_with_warmup`
        - Learning Rate: `3e-5`
        - Batch Size: `1`
        - Epoch: `2`
        - Gradient Accumulation Step: `40`
      - Question Answering:
        - Optimization Algorithm: `transformers.AdamW`
        - Scheduler: `transformers.optimization.get_linear_schedule_with_warmup`
        - Learning Rate: `3e-5`
        - Batch Size: `8`
        - Epoch: `2`
        - Gradient Accumulation Step: `2`
  - **the performance of this model v.s. BERT**
    - 在Context Selection的task上，轉確率為40%，由於是7選1，其實這樣的表現只比隨機亂猜好一點。
    - 在Question Answering上，有2%的命中率，會表現這麼差的原因是因為訓練資料太少，而原本使用pre-trained model有很多contextualized 的knowledge，現在沒有後就造成巨幅退步。

```
{"count": 3526, "em": 0.012548423842356, "f1": 0.0337884272748309}
```

    - 以上是overall的Performance，可以看到整體退步了非常多。

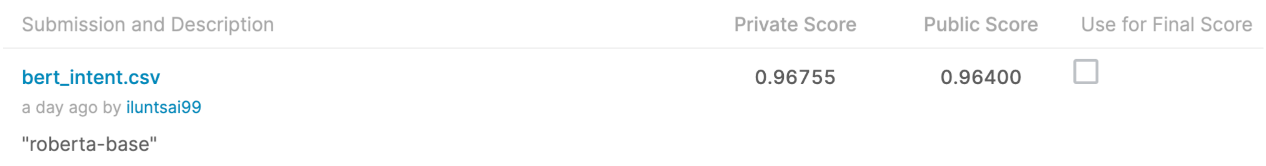
5. Compare with different configurations (1% + Bonus 1%)

- **Train a BERT-based model on HW1 dataset and describe**
- **Intent Classification:**
    - **your model**
        - 使用`"roberta-base"`這個pre-trained model做`RobertaForSequenceClassification`的task，定義了`num_labels=150`（150種intent）。
        - Configuration:

```
"architectures": [
  "RobertaForSequenceClassification"
],
"attention_probs_dropout_prob": 0.1,
"bos_token_id": 0,
"eos_token_id": 2,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 768,
"initializer_range": 0.02,
"intermediate_size": 3072,
"layer_norm_eps": 1e-05,
"max_position_embeddings": 514,
"model_type": "roberta",
"num_attention_heads": 12,
"num_hidden_layers": 12,
"pad_token_id": 1,
"type_vocab_size": 1,
"vocab_size": 50265
```

❗ 注意原本config還有兩個dictionary（id2label和label2id），但因為要歸類成150種佔太多空間就省略不放了。

    - **performance of your model.**



Submission and Description	Private Score	Public Score	Use for Final Score
bert_intent.csv a day ago by iluntsai99 "roberta-base"	0.96755	0.96400	☐

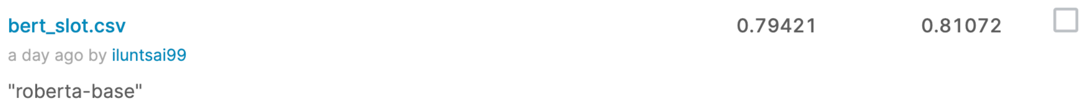
        - 可以看出使用BERT讓performance從92%進步到近97%，推測這是因為BERT的contextualize embedding比原本使用的GLOVE好很多。
    - **the loss function you used.**
        - 選取的intent(label)對ground truth做`Cross-Entropy Loss`。
    - **The optimization algorithm (e.g. Adam), learning rate and batch size.**

        - Optimization Algorithm: `transformers.AdamW`
        - Scheduler: `transformers.optimization.get_linear_schedule_with_warmup`
        - Learning Rate: `5e-5`
        - Batch Size: `64`
        - Epoch: `5`
        - Gradient Accumulation Step: `2`
- **Slot Tagging:**
    - **your model**
        - 使用`"roberta-base"`這個pre-trained model做`RobertaForTokenClassification`的task，定義了`num_labels=9`（9個tag）。
        - Configuration:

```
"architectures": [
  "RobertaForTokenClassification"
],
"attention_probs_dropout_prob": 0.1,
"bos_token_id": 0,
"eos_token_id": 2,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 768,
"initializer_range": 0.02,
"intermediate_size": 3072,
"layer_norm_eps": 1e-05,
"max_position_embeddings": 514,
"model_type": "roberta",
"num_attention_heads": 12,
"num_hidden_layers": 12,
"pad_token_id": 1,
"type_vocab_size": 1,
"vocab_size": 50265
```

❗ 跟intent的task一樣，原本還有兩個dictionary（id2label和label2id），分成9種NER，這邊一樣省略不放。

    - **performance of your model.**



bert_slot.csv 0.79421 0.81072
a day ago by iluntsai99
"roberta-base"

        - 原本使用GRU就可以達到82.3%，使用BERT反而讓performance從82.3%退步到近81%。
        - 嘗試使用large的model想增進performance反而退步更多，推測是由於提供的slot data不夠多，無法有效地fine-tune pre-trained好的model（原本model的參數太dominant）。

- **the loss function you used.**
  - 對每一個字選取的tag(label)做`Cross-Entropy Loss`，並加起來作為total loss。
- **The optimization algorithm (e.g. Adam), learning rate and batch size.**
  - Optimization Algorithm: `transformers.AdamW`
  - Scheduler: `transformers.optimization.get_linear_schedule_with_warmup`
  - Learning Rate: `5e-5`
  - Batch Size: `32`
  - Epoch: `5`
  - Gradient Accumulation Step: `1`