Electra fine-tuning 問答任務實作

1092950 林佳笙

目錄

實作目標

利用 electra 模型練習 fine-tuning，對此模型多任務面向中，針對 Question Answering(QA)任務進行訓練實作。

以 electra-small-discriminator 作為預訓練模型，使用 SquAD 資料集取其中 10000 筆作為訓練使用。

模型介紹

1. 簡介

ELECTRA: PRE-TRAINING TEXT ENCODERS AS DISCRIMINATORS RATHER THAN GENERATORS

Kevin Clark
Stanford University
kevclark@cs.stanford.edu

Minh-Thang Luong
Google Brain
thangluong@google.com

Quoc V. Le
Google Brain
qvl@google.com

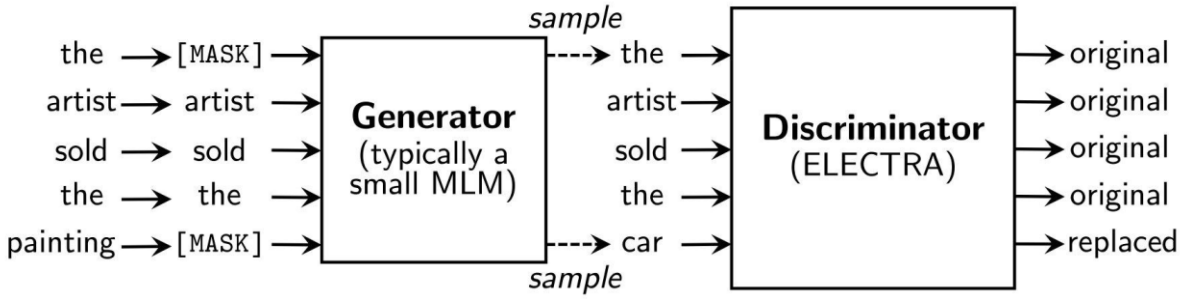
Christopher D. Manning
Stanford University & CIFAR Fellow
manning@cs.stanford.edu

Electra 為一 Generator-Discriminator 架構的模型，並提出了一種新的預訓練任務—replaced token detection(RTD)作為訓練，透過替換部分 token 破壞輸入後，預測被替換的 token。
而其中一大優勢為項較 bert 與 GPT 模型，降低了訓練的成本需求。

目前在 electra 的 github 已公布了三種大小的預訓練模型：

Model	Layers	Hidden Size	Params	GLUE score (test set)	Download
ELECTRA-Small	12	256	14M	77.4	link
ELECTRA-Base	12	768	110M	82.7	link
ELECTRA-Large	24	1024	335M	85.2	link

2. 模型架構



作者提供一個新的框架 Generator-Dicriminator 配合 RTD 進行訓練，而在後續的實驗證實此架構下的效率與成果來得更加優秀。

而相對於相似的 GAN 架構，作者提出了仍有不少的差別：

	ELECTRA	GAN
輸入	真實文本	隨機噪聲
目標	生成器學習語言模型，判別器學習區分真假文本	生成器盡可能欺騙判別器，判別器盡量區分真假圖片
反向傳播	梯度無法從 D 傳到 G	梯度可以從 D 傳到 G
特殊情況	生成出了真實文本，則標記為正例	生成的都是負例（假圖片）

3. 實驗數據

實驗數據中，作者為提升預訓練效率，做了單個 GPU 即可訓練的 bert-small、electra-small，比對如下：

Model	Train / Infer FLOPs	Speedup	Params	Train Time + Hardware	GLUE
ELMo	3.3e18 / 2.6e10	19x / 1.2x	96M	14d on 3 GTX 1080 GPUs	71.2
GPT	4.0e19 / 3.0e10	1.6x / 0.97x	117M	25d on 8 P6000 GPUs	78.8
BERT-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100 GPU	75.1
BERT-Base	6.4e19 / 2.9e10	1x / 1x	110M	4d on 16 TPUv3s	82.2
ELECTRA-Small	1.4e18 / 3.7e9	45x / 8x	14M	4d on 1 V100 GPU	79.9

兩者以僅 14M 的參數量與更快的效率達到了相近的成效。

資料集介紹

SQuAD

SQuAD (Stanford Question Answering Dataset)

Introduced by Rajpurkar et al. in SQuAD: 100,000+ Questions for Machine Comprehension of Text

The **Stanford Question Answering Dataset** (**SQuAD**) is a collection of question-answer pairs derived from Wikipedia articles. In SQuAD, the correct answers of questions can be any sequence of tokens in the given text. Because the questions and answers are produced by humans through crowdsourcing, it is more diverse than some other question-answering datasets. SQuAD 1.1 contains 107,785 question-answer pairs on 536 articles. SQuAD2.0 (open-domain SQuAD, SQuAD-Open), the latest version, combines the 100,000 questions in SQuAD1.1 with over 50,000 un-answerable questions written adversarially by crowdworkers in forms that are similar to the answerable ones.

此資料集為一個閱讀理解的資料集，主要收錄涉及維基百科上文章的問答，而每個問題的答案都是相應閱讀段落的文本或者無法回答，有超過 500 篇與 10 萬以上問答。SQuAD 資料集分為 1.1 與 2.0，後者在原先多項一問一答中，增添了更多通用的答句在同一問題上。

實驗方法

實驗設備

```
+-----------------------------------------------------------------------------+
| NVIDIA-SMI 531.79                 Driver Version: 531.79       CUDA Version: 12.1     |
|-------------------------------------+----------------------+----------------------+
| GPU  Name                      TCC/WDDM | Bus-Id        Disp.A | Volatile Uncorr. ECC |
| Fan  Temp  Perf            Pwr:Usage/Cap|         Memory-Usage | GPU-Util  Compute M. |
|                                     |                      |               MIG M. |
|=====================================+======================+======================|
|   0  NVIDIA GeForce RTX 2060       WDDM | 00000000:01:00.0 Off |                  N/A |
| N/A   41C    P5               10W /  N/A|    198MiB /  6144MiB |     15%      Default |
|                                     |                      |                  N/A |
+-------------------------------------+----------------------+----------------------+
```

Torch:2.1.2

實驗參數

超參數皆無變動

```
"attention_probs_dropout_prob": 0.1,
"classifier_dropout": null,
"embedding_size": 128,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 256,
"initializer_range": 0.02,
"intermediate_size": 1024,
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "electra",
"num_attention_heads": 4,
"num_hidden_layers": 12,
"pad_token_id": 0,
"position_embedding_type": "absolute",
"summary_activation": "gelu",
"summary_last_dropout": 0.1,
"summary_type": "first",
"summary_use_proj": true,
"transformers_version": "4.32.1",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 30522
```

實驗步驟

1. 資料集載入

   1-1 透過 hugging face 載入 SQuAD1.1 資料集，本實驗僅取 10000 筆。

```
from datasets import load_dataset

squad = load_dataset("squad", split="train[:10000]")
```

   1-2 以 0.8、0.2 比例分割訓練集與測試集(驗證集)

```
squad = squad.train_test_split(test_size=0.2)
```

   1-3 資料集格式如下：

```
squad["train"][0]
```

✓ 0.0s

```
{'id': '56bec29b3aeaaa14008c9382',
 'title': 'Beyoncé',
 'context': 'In 2006, Beyoncé introduced her all-female
 'question': 'The Mamas first appearance was when?',
 'answers': {'text': ['2006'], 'answer_start': [3]}}
```

2. 資料前處裡

   2-1 創建 QA task 格式的資料，以用於後續訓練使用。

```
max_length = 384
stride = 128

def preprocess_function(examples):
    questions = [q.strip() for q in examples["question"]]
    inputs = tokenizer(
        questions,
        examples["context"],
        max_length=max_length,
        stride=stride,
        truncation="only_second",
        return_offsets_mapping=True,
        padding="max_length",
    )

    offset_mapping = inputs.pop("offset_mapping")
    answers = examples["answers"]
    start_positions = []
    end_positions = []

    for i, offset in enumerate(offset_mapping):
        answer = answers[i]
        start_char = answer["answer_start"][0]
        end_char = answer["answer_start"][0] + len(answer["text"][0])
        sequence_ids = inputs.sequence_ids(i)

        # Find the start and end of the context
        idx = 0
        while sequence_ids[idx] != 1:
            idx += 1
```

2-1 用 map 將訓練集資料轉換成想要的格式，格式如下圖。

```
train_squad = squad["train"].map(preprocess_function, batched=True, remove_columns=squad["train"].column_names)

len(squad["train"]), len(train_squad)
✓ 6.6s
```

```
(8000, 8000)
```

```
train_squad
✓ 0.0s
```

```
Dataset({
    features: ['input_ids', 'token_type_ids', 'attention_mask', 'start_positions', 'end_positions'],
    num_rows: 8000
})
```

QA 資料的格式主要為[CLS]問句[SEP]答句[SEP][PAD].....(用[PAD]填滿統一格式，本實驗最大長度設為 384)，因此需要記錄到兩個問句與答句的位置才能取出正確問答句。(如下圖)

```
[CLS] the mamas first appearance was when? [SEP] in 2006, beyonce introduced her all - female tour band suga mama ( also the name
of a song in b'day ) which includes bassists, drummers, guitarists, horn players, keyboardists and percussionists. her background
singers, the mamas, consist of montina cooper - donnell, crystal collins and tiffany monique riddick. they made their debut
appearance at the 2006 bet awards and re - appeared in the music videos for " irreplaceable " and " green light ". the band have
supported beyonce in most subsequent live performances, including her 2007 concert tour the beyonce experience, 2009 – 2010 i am...
world tour and 2013 – 2014 the mrs. carter show world tour. [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
```

2-2 驗證集也需要，用於驗證答案。

```
validation_squad = squad["test"].map(
    preprocess_validation_examples,
    batched=True,
    remove_columns=squad["test"].column_names,
)
len(squad["test"]), len(validation_squad)
✓ 1.9s
```

```
(2000, 2031)
```

```
validation_squad
✓ 0.0s
```

```
Dataset({
    features: ['input_ids', 'token_type_ids', 'attention_mask', 'offset_mapping', 'example_id'],
    num_rows: 2031
})
```

3. 訓練流程

3-1 創建訓練集與驗證集的 dataloader，並在此處調整 batch_size 觀察差異

```
from torch.utils.data import DataLoader
from transformers import default_data_collator

train_squad.set_format("torch")
validation_set = validation_squad.remove_columns(["example_id", "offset_mapping"])
validation_set.set_format("torch")

train_dataloader = DataLoader(
    train_squad,
    shuffle=True,
    collate_fn=default_data_collator,
    batch_size=8,
)
eval_dataloader = DataLoader(
    validation_set, collate_fn=default_data_collator, batch_size=8
)
```

0.0s

3-2 透過 hugging face 載入 QA task 初始參數的預訓練模型

```
model = AutoModelForQuestionAnswering.from_pretrained(model_name)
```

✓ 1.3s

```
Some weights of ElectraForQuestionAnswering were not initialized from the
and are newly initialized: ['qa_outputs.bias', 'qa_outputs.weight']
You should probably TRAIN this model on a down-stream task to be able to
```

3-3 選擇優化器，本實驗選擇的是 Adam

```
from torch.optim import AdamW

optimizer = AdamW(model.parameters(), lr=2e-5)
```

3-4 hugging face 中的加速器，優化訓練的設置

```
from accelerate import Accelerator

accelerator = Accelerator()
model, optimizer, train_dataloader, eval_dataloader = accelerator.prepare(
    model, optimizer, train_dataloader, eval_dataloader
)
```

0.0s

3-5 實驗使用線性的策略來調整 learning rate，此處也設定實驗 epoch 為 20

```
from transformers import get_scheduler

num_train_epochs = 20
num_update_steps_per_epoch = len(train_dataloader)
num_training_steps = num_train_epochs * num_update_steps_per_epoch

lr_scheduler = get_scheduler(
    "linear",
    optimizer=optimizer,
    num_warmup_steps=0,
    num_training_steps=num_training_steps,
)
```

3-6 自訂驗證方法，SQuAD 主要以 Exact_match 為評估分數，而 f1 score 也是在 QA 任務中用來參考使用的評估分數，故實驗使用此兩種分數作為評判。
直接透過 hugging face 載入 SQuAD 的預設評估方法。

```
import evaluate

metric = evaluate.load("squad")
```

此處參考 hugging face 對 QA task 設計的範例評估函式來做使用

```
def compute_metrics(start_logits, end_logits, features, examples):
    example_to_features = collections.defaultdict(list)
    for idx, feature in enumerate(features):
        example_to_features[feature["example_id"]].append(idx)

    predicted_answers = []
    for example in tqdm(examples):
        example_id = example["id"]
        context = example["context"]
        answers = []

        # Loop through all features associated with that example
        for feature_index in example_to_features[example_id]:
            start_logit = start_logits[feature_index]
            end_logit = end_logits[feature_index]
            offsets = features[feature_index]["offset_mapping"]

            start_indexes = np.argsort(start_logit)[-1 : -n_best - 1 : -1].tolist()
            end_indexes = np.argsort(end_logit)[-1 : -n_best - 1 : -1].tolist()
            for start_index in start_indexes:
                for end_index in end_indexes:
                    # Skip answers that are not fully in the context
                    if offsets[start_index] is None or offsets[end_index] is None:
                        continue
                    # Skip answers with a length that is either < 0 or > max_answer_length
                    if (
                        end_index < start_index
                        or end_index - start_index + 1 > max_answer_length
                    ):
                        continue
```

生成的分數即為 EM(exact_match)與 f1。

```
compute_metrics(start_logits, end_logits, eval_set, small_eval_set)
✓ 0.4s
100% 100/100 [00:00<00:00, 421.12it/s]
{'exact_match': 84.0, 'f1': 92.0308608058608}
```

3-7 訓練，並手動在過程記錄 loss

```python
from tqdm.auto import tqdm
import torch

progress_bar = tqdm(range(num_training_steps))
train_history_score = list()
batch_loss = 0

for epoch in range(num_train_epochs):
    # Training
    model.train()
    for step, batch in enumerate(train_dataloader):
        outputs = model(**batch)
        loss = outputs.loss
        accelerator.backward(loss)

        optimizer.step()
        lr_scheduler.step()
        optimizer.zero_grad()
        progress_bar.update(1)

        # print(loss)
        # print(loss.item())
        batch_loss += loss.item()
    batch_loss /= len(train_dataloader.dataset)
```

3-8 訓練中的驗證，最後將訓練的評估數據:loss、exact_match、f1 保存

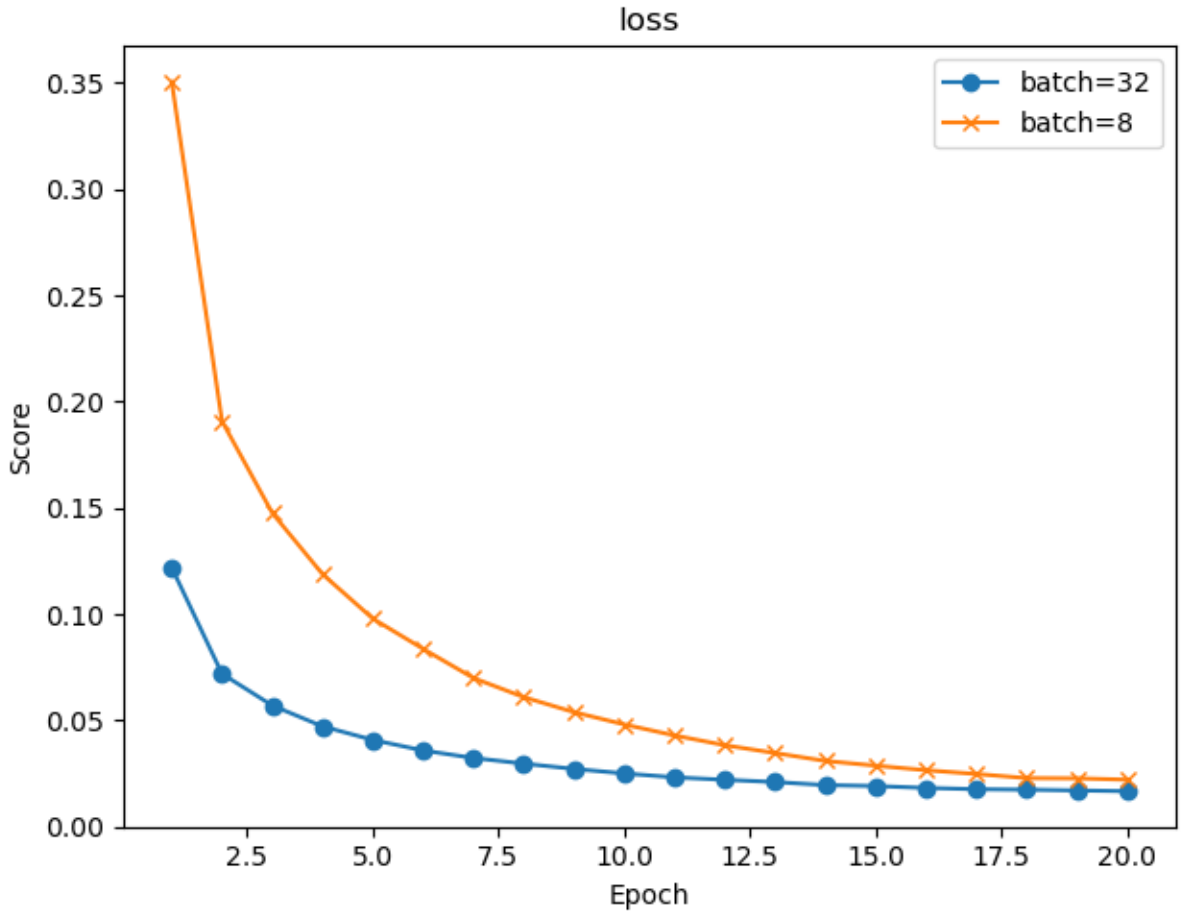
```
# Evaluation
model.eval()
start_logits = []
end_logits = []
accelerator.print("Evaluation!")
for batch in tqdm(eval_dataloader):
    with torch.no_grad():
        outputs = model(**batch)

    start_logits.append(accelerator.gather(outputs.start_logits).cpu().numpy())
    end_logits.append(accelerator.gather(outputs.end_logits).cpu().numpy())

start_logits = np.concatenate(start_logits)
end_logits = np.concatenate(end_logits)
start_logits = start_logits[: len(validation_squad)]
end_logits = end_logits[: len(validation_squad)]

metrics = compute_metrics(
    start_logits, end_logits, validation_squad, squad["test"]
)
print(f"epoch {epoch}:", metrics)
train_history_score.append([batch_loss, metrics["exact_match"], metrics["f1"]])
```

實驗結果

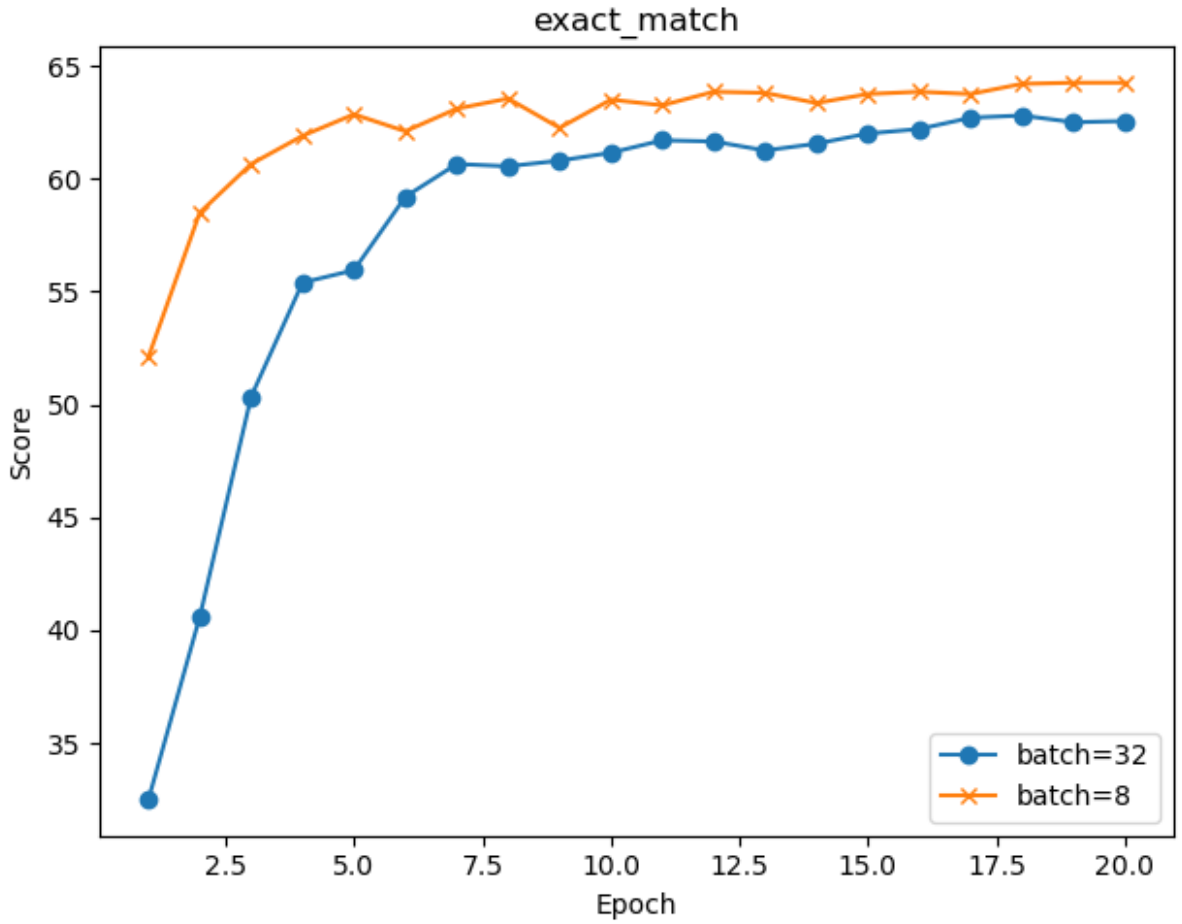
- 10000 切割為 0.8 訓練及與 0.2 驗證集
- 20 epoch
- 添加使用 Adam 優化器，leanring rate = 2e-5
- 訓練過程中比較了 batch 為 8 與 batch 為 32 時的差異。

實驗數據

Loss：開始 Batch=8 明顯較高，但整體在後續與 batch=32 趨於相近，可說明 Batch=8 在收斂速度上是較大的。

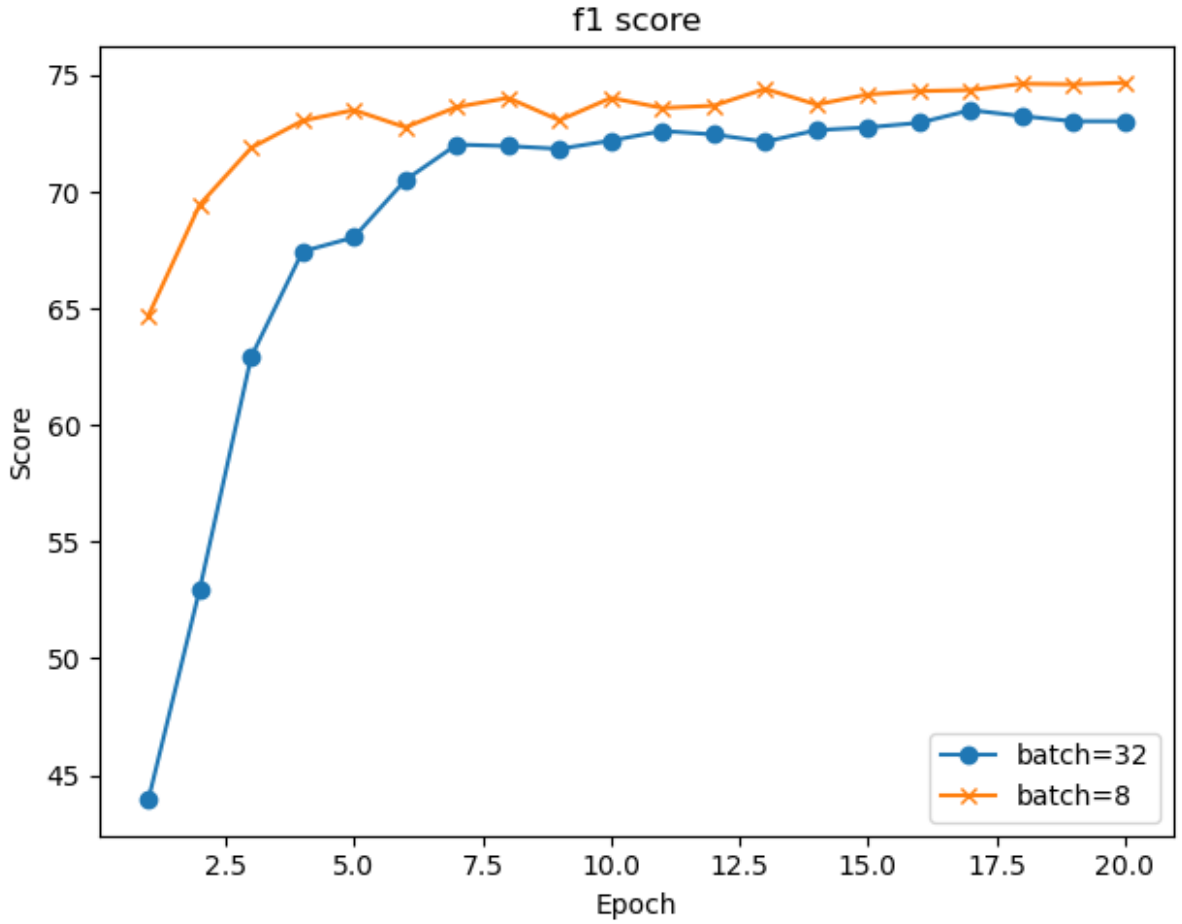


Exact_match:從下圖可以評估整體上 batch=8 在訓練表現上比 batch=32 來的更好，較小的 batch 使訓練調整的成效更佳，而和原論文作者的實際分數相比低了約 10 個百分點
原作者：75.8 本實驗： batch=32：62.55 batch=8：64.25。
以上數據差異有多項因素可能導致，如：使用資料集訓練的大小、epoch 量、優化器的調整。但個人認為以資料集訓練的使用大小影響為最大。



	CoLA	SST	MRPC	STS	QQP	MNLI	QNLI	RTE	SQuAD 1.1	SQuAD 2.0
Metrics	MCC	Acc	Acc	Spearman	Acc	Acc	Acc	Acc	EM	EM
ELECTRA-Large	69.1	96.9	90.8	92.6	92.4	90.9	95.0	88.0	89.7	88.1
ELECTRA-Base	67.7	95.1	89.5	91.2	91.5	88.8	93.2	82.7	86.8	80.5
ELECTRA-Small	57.0	91.2	88.0	87.5	89.0	81.3	88.4	66.7	75.8	70.1
ELECTRA-Small-OWT	56.8	88.3	87.4	86.8	88.3	78.9	87.9	68.5	--	--

F1 score: f1 分數與 exact_match 圖形表現上差異不大，以目前訓練的 epoch 來看，兩者在更多的訓練次數下或許仍有些微增長的空間。



結論

這次期末雖然遇到了多種困難，特別是使用 hugging face 的支援來完整 fine-tune 一個模型，針對某項任務時應如何設計並在後續微調，以及最後如何將數據型態的訓練成果製圖顯示。原先在 ppt 上訓練過程中使用了 generator 是錯誤的方式，也是在多次瞭解 electra 模型的概念後才明瞭。

但是相比花費的時間相比，得到的成果相當的多，透過互相比較更加了解一些超參數的改動意義，以及 hugging face 中針對訓練的 Tainer，在有許多客製化要求或是變動時，熟悉與否以及便利性的差別也就體現出來了。

回歸本次實驗，讓我完整地從基本認識了微調實際上的步驟與部分考量內容，但也因為使用 hugging face 的 API，使得我對預訓練、模型內部隱藏層變動的了解仍不夠詳盡，希望後續有機會能完善，以更加明白實驗的流程預設計可以有哪些考量。

參考資料

[1] Electra: https://github.com/google-research/electra

[2] Hugging Face – electra-small-discriminator: https://huggingface.co/google/electra-small-discriminator

[3] SQuAD: https://rajpurkar.github.io/SQuAD-explorer/