Project Description

Build a BERT-based model which returns “an answer”, given a user question and a passage which includes the answer of the question. For this question answering task, we will use the SQuAD 2.0 dataset. We will start with the BERT-base pretrained model “bert-base-uncased” and fine-tune it to have a question answering task.

Question 3

Load SCuAD 2.0 dataset from drive

```
import io
import os
from google.colab import drive
import pandas as pd
import numpy as np
import json
import sys

drive.mount('/content/drive',force_remount=True)

sys.path.append('/content/drive/My Drive/')

!cp -r "/content/drive/My Drive/train-v2.0.json" '/content/'
!cp -r "/content/drive/My Drive/dev-v2.0.json" '/content/'
!cp -r "/content/drive/My Drive/utils_squad.py" '/content/'
!cp -r "/content/drive/My Drive/utils_squad_evaluate.py" '/content/'

train_file = '/content/train-v2.0.json'
validation_file = '/content/dev-v2.0.json'

with open(train_file) as f:
    raw_train_data = json.load(f)
with open(validation_file) as f:
    raw_val_data = json.load(f)
```

```
Mounted at /content/drive
```

Install transformers

```
%%capture
!pip install transformers
```

Create the tokenizer

```
import torch
from transformers import BertTokenizer, BertForSequenceClassification
from transformers import BertForQuestionAnswering
from tokenizers import BertWordPieceTokenizer

# Load pre-trained model tokenizer (vocabulary)
slow_tokenizer = BertTokenizer.from_pretrained("bert-base-uncased")
save_path = "bert_base_uncased/"
if not os.path.exists(save_path):
    os.makedirs(save_path)
slow_tokenizer.save_pretrained(save_path)

# Load the fast tokenizer from saved file
tokenizer = BertWordPieceTokenizer("bert_base_uncased/vocab.txt", lowercase=True)
max_len = 384
```

```
Downloading: 100%                    232k/232k [00:00<00:00, 308kB/s]
```

Create a class to save the data form sQuAD

```
class SquadSample:

    def __init__(self, context, question, basic_answer, more_answers, start_idx):
```

```
        self.context = context
        self.question = question
        self.basic_answer = basic_answer
        self.more_answers = more_answers
        self.start_idx = start_idx
        self.end_idx = None
        self.start_idx_token = start_idx
        self.end_idx_token = None
        self.offsets = None
        self.input_ids = None
        self.attention_mask = None
        self.token_type_ids = None
        self.validExample = True

    def preprocess(self): # function to preprocess data
       # Clean context, answer and question
        self.context = " ".join(str(self.context).split())
        self.question = " ".join(str(self.question).split())

        contextTokenizer =  tokenizer.encode(self.context)

        if self.basic_answer is not None: # in case we have an answer

          self.basic_answer = " ".join(str(self.basic_answer).split())
          #Calculate end_idx
          self.end_idx = self.start_idx + len(self.basic_answer)
          if (self.end_idx >=len(self.context)):
              self.validExample= False
              return

          #find characters of context that are part of answer
          is_part_of_answer = [0]*len(self.context)
          for i in range (self.start_idx, self.end_idx):
              is_part_of_answer[i] = 1


          #find index of token that corresponds to start and the end of the answer
          answer_id_token=[]
          for idx, (start,end) in enumerate(contextTokenizer.offsets):
              if (sum(is_part_of_answer[start:end]) >0 ):
                  answer_id_token.append(idx)
          #data to predict
          if len(answer_id_token) == 0 :
              self.validExample=False
              return
          self.start_idx_token = answer_id_token[0]
          self.end_idx_token = answer_id_token[-1]
        self.offsets = contextTokenizer.offsets

        # work on question
        questionTokinizer  = tokenizer.encode(self.question)

        #Create model's inputs
        self.input_ids = contextTokenizer.ids + questionTokinizer.ids[1:]
        self.attention_mask = [1] * len (self.input_ids)
        self.token_type_ids = [0] * len(contextTokenizer.ids) + [1]*len(questionTokinizer.ids[1:])

        # fix padding
        padding_length = max_len - len(self.input_ids)
        if padding_length > 0:
            self.input_ids = self.input_ids + ([0] * padding_length)
            self.attention_mask = self.attention_mask + ([0] * padding_length)
            self.token_type_ids = self.token_type_ids + ([0] * padding_length)
        elif padding_length < 0:
            self.validExample= False
            return
```

Function that helps save data from json files

```
def create_squad_examples(raw_data):
    squad_examples = []
    for item in raw_data["data"]:
        for para in item["paragraphs"]:
            context = para["context"]
            for qa in para["qas"]:
                question = qa["question"]
                if qa["answers"]:
                  answer_text = qa["answers"][0]["text"]
                  all_answers = [_["text"] for _ in qa["answers"]]
                  start_char_idx = qa["answers"][0]["answer_start"]
                  #context, question, basic_answer, more_answers, start_idx
                  squad_eg = SquadSample(context,question, answer_text, all_answers, start_char_idx)
```

```
                else:
                  squad_eg = SquadSample(context,question, None, None, None) #context, question
                squad_eg.preprocess()
                squad_examples.append(squad_eg)
    return squad_examples
```

Function that create ttwo dictionaries, one for the input and one for the target

```
def create_inputs_targets(squad_examples):
    dataset_dict = {
        "input_ids" : [],
        "attention_mask" : [],
        "token_type_ids" : [],
        "start_idx_token" : [],
        "end_idx_token" : []
    }
    for item in squad_examples:
        if item.validExample is True:
            for key in dataset_dict:
                dataset_dict[key].append(getattr(item, key))
    for key in dataset_dict:
        dataset_dict[key] = np.array(dataset_dict[key],dtype=np.float16)
    x = [dataset_dict["input_ids"], dataset_dict["attention_mask"], dataset_dict["token_type_ids"]]
    y = [dataset_dict["start_idx_token"], dataset_dict["end_idx_token"]]
    return x, y
```

```
data = create_squad_examples(raw_train_data)  # save the data for the training set

val_data = create_squad_examples(raw_val_data)  # save the data for the validation set
```

Check what is saved

```
train_data = pd.DataFrame.from_records([vars(line) for line in data])
train_data[["context","question","basic_answer"]].head()
```

	context	question	basic_answer
0	Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ b...	When did Beyonce start becoming popular?	in the late 1990s
1	Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ b...	What areas did Beyonce compete in when she was...	singing and dancing
2	Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ b...	When did Beyonce leave Destiny's Child and bec...	2003
3	Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ b...	In what city and state did Beyonce grow up?	Houston, Texas
4	Beyoncé Giselle Knowles-Carter (/biːˈjɒnseɪ/ b...	In which decade did Beyonce become famous?	late 1990s

```
x_train, y_train = create_inputs_targets(data)  # split the training data to input and target

x_eval, y_eval = create_inputs_targets(val_data)  # split the validation data to input and target
```

```
doc_stride = 64
max_seq_length = 128
max_query_length = 32
batch_size = 16
```

Create Datasets

```
import torch
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler

# Convert to Tensors and build dataset
train_data = TensorDataset(torch.tensor(x_train[0], dtype=torch.int64),
                           torch.tensor(x_train[1], dtype=torch.float),
                           torch.tensor(x_train[2], dtype=torch.int64),
                           torch.tensor(y_train[0], dtype=torch.int64),
                           torch.tensor(y_train[1], dtype=torch.int64))

train_sampler = RandomSampler(train_data)
train_dataloader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size)
```

```
# Convert to Tensors and build dataset
train_sampler = RandomSampler(train_data)
train_data_loader = DataLoader(train_data, sampler=train_sampler, batch_size=batch_size)
eval_data = TensorDataset(torch.tensor(x_eval[0], dtype=torch.int64),
                          torch.tensor(x_eval[1], dtype=torch.float),
```

```
                          torch.tensor(x_eval[2], dtype=torch.int64),
                          torch.tensor(y_eval[0], dtype=torch.int64),
                          torch.tensor(y_eval[1], dtype=torch.int64))

eval_sampler = SequentialSampler(eval_data)
validation_dataloader = DataLoader(eval_data, sampler=eval_sampler, batch_size=batch_size)
```

Check for GPU availability

```
import torch
# First checking if GPU is available
train_on_gpu=torch.cuda.is_available()

if(train_on_gpu):
    print('Training on GPU.')
    device = 'cuda'
else:
    print('No GPU available, training on CPU.')
    device = 'cpu'
```

```
Training on GPU.
```

Initialize model

```
model = BertForQuestionAnswering.from_pretrained('bert-base-uncased').to(device=device)
param_optimizer = list(model.named_parameters())
```

```
Downloading: 100%                    433/433 [00:16<00:00, 25.6B/s]

Downloading: 100%                    440M/440M [00:15<00:00, 27.5MB/s]

Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForQuestionAnswering: ['cls.predictions
- This IS expected if you are initializing BertForQuestionAnswering from the checkpoint of a model trained on another task or with an
- This IS NOT expected if you are initializing BertForQuestionAnswering from the checkpoint of a model that you expect to be exactly
Some weights of BertForQuestionAnswering were not initialized from the model checkpoint at bert-base-uncased and are newly initialize
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
no_decay = ['bias', 'gamma', 'beta']
optimizer_grouped_parameters = [
    {'params': [p for n, p in param_optimizer if not any(nd in n for nd in no_decay)],
     'weight_decay_rate': 0.01},
    {'params': [p for n, p in param_optimizer if any(nd in n for nd in no_decay)],
     'weight_decay_rate': 0.0}
]

optimizer = torch.optim.Adam(lr=1e-5, betas=(0.9, 0.98), eps=1e-9, params=optimizer_grouped_parameters)
```

Training model

Function for text normalization

```
import string
import re

def normalize_text(text):
    text = text.lower()
    text = "".join(ch for ch in text if ch not in set(string.punctuation))
    regex = re.compile(r"\b(a|an|the)\b", re.UNICODE)
    text = re.sub(regex, " ", text)
    text = " ".join(text.split())
    return text
```

```
epochs = 1

for epoch in range(1, epochs + 1):
    # ========================================== TRAINING ==============================================================
    print("Training epoch ", str(epoch))

    model.train()
    tr_loss = 0
    nb_tr_steps = 0
    for step, batch in enumerate(train_dataloader):
        batch = tuple(t.to(device) for t in batch)
```

```
        inputs = {'input_ids':       batch[0],
                  'attention_mask':  batch[1],
                  'token_type_ids':  batch[2],
                  'start_positions': batch[3],
                  'end_positions':   batch[4]}

        optimizer.zero_grad()

        outputs = model(**inputs)
        loss = outputs[0]

        loss.backward()
        optimizer.step()
        tr_loss += loss.item()
        nb_tr_steps += 1

    print(f"\nTraining loss={tr_loss / nb_tr_steps:.4f}")

    # ========================================== VALIDATION ==========================================================
    model.eval()
    currect_query = 0
    correct_ans = 0
    valid_examples = [x for x in val_data if x.validExample is True]
    for batch in validation_dataloader:
        batch = tuple(t.to(device) for t in batch)

        input_ids, attention_mask, token_type_ids, start_positions, end_positions = batch

        with torch.no_grad():
          start_logits, end_logits =  model(input_ids=input_ids,
                                            attention_mask=attention_mask,
                                            token_type_ids=token_type_ids, return_dict=False)

          pred_start, pred_end = start_logits.detach().cpu().numpy(), end_logits.detach().cpu().numpy()

        for idx, (start, end) in enumerate(zip(pred_start, pred_end)):
            squad_eg = valid_examples[currect_query]
            currect_query += 1
            offsets = squad_eg.offsets
            start = np.argmax(start)
            end = np.argmax(end)
            if start >= len(offsets):
                continue
            pred_char_start = offsets[start][0]
            if end < len(offsets):
                pred_char_end = offsets[end][1]
                pred_ans = squad_eg.context[pred_char_start:pred_char_end]
            else:
                pred_ans = squad_eg.context[pred_char_start:]
            normalized_pred_ans = normalize_text(pred_ans)
            normalized_true_ans = [normalize_text(x) for x in squad_eg.more_answers]
            if normalized_pred_ans in normalized_true_ans:
                correct_ans += 1
    acc = correct_ans / len(y_eval[0])

    print(f"\nAccuracy score={acc:.2f}\n")
    print("----------------------------------------")
```

```
Training epoch  1

Training loss=1.9051

Accuracy score=0.57

----------------------------------------
```

Testing model

```
# ========================================== TESTING ==========================================================
data = {"data":
    [
        {"title": "Tesla's Biography",
         "paragraphs": [
             {
                 "context": "Nikola Tesla was a Serbian-American inventor, electrical engineer, mechanical engineer, "
                            "and futurist best known for his contributions to the design of the modern alternating "
                            "current (AC) electricity supply system. Born and raised in the Austrian Empire,Tesla "
                            "studied engineering and physics in the 1870s without receiving a degree. In 1884, he "
                            "moved to United States. In 1887, Tesla developed an induction motor that ran on alternating "
                            "current (AC), a power system format that was rapidly expanding in Europe and the United "
                            "States because of its advantages in long-distance, high-voltage transmission. Tesla became "
```

```
                            "a vegetarian in his later years, living on only milk, bread, honey, and vegetable juices. "
                            "On 7 January 1943, at the age of 86, Tesla died alone in Room 3327 of the Hotel New Yorker. "
                            "Tesla wrote a number of books and articles for magazines and journals. Among his books are "
                            "My Inventions: The Autobiography of Nikola Tesla, compiled and edited by Ben Johnston "
                            "in 1983 from a series of 1919 magazine articles by Tesla which were republished in 1977. "
                            "Tesla's legacy has endured in books, films, radio, TV, music, live theater, comics, and "
                            "video games. The impact of the technologies invented or envisioned by Tesla is a recurring "
                            "theme in several types of science fiction. ",
                    "qas": [
                        {"question": "When did Tesla become a vegetarian?",
                         "id": "Q1",
                         "answers":""
                         },
                        {"question": "When did Tesla move to United States ?",
                         "id": "Q2",
                         "answers":""
                         },
                        {"question": "What year did Tesla die?",
                         "id": "Q3",
                         "answers":""
                         },
                        {"question": "Who edited the book My Inventions: The Autobiography of Nikola Tesla?",
                         "id": "Q4",
                         "answers":""
                         },
                        {"question": "In what age did Tesla died?",
                         "id": "Q5",
                         "answers":""
                         },
                        {"question": "Who developed an induction motor?",
                         "id": "Q6",
                         "answers":""
                         },
                        {"question": "Where Tesla was born?",
                         "id": "Q7",
                         "answers":""
                         },
                        {"question": "What did Tesla study?",
                         "id": "Q8",
                         "answers":""
                         },
                    ]}]}]}
```


```
model.eval()
test_samples = create_squad_examples(data)
x_test, _ = create_inputs_targets(test_samples)
pred_start, pred_end = model(torch.tensor(x_test[0], dtype=torch.int64, device=device),
                             torch.tensor(x_test[1], dtype=torch.float, device=device),
                             torch.tensor(x_test[2], dtype=torch.int64, device=device), return_dict=False)
pred_start, pred_end = pred_start.detach().cpu().numpy(), pred_end.detach().cpu().numpy()
for idx, (start, end) in enumerate(zip(pred_start, pred_end)):
    test_sample = test_samples[idx]
    offsets = test_sample.offsets
    start = np.argmax(start)
    end = np.argmax(end)
    pred_ans = None
    if start >= len(offsets):
        continue
    pred_char_start = offsets[start][0]
    if end < len(offsets):
        pred_ans = test_sample.context[pred_char_start:offsets[end][1]]
    else:
        pred_ans = test_sample.context[pred_char_start:]
    print("Q: " + test_sample.question)
    print("A: " + pred_ans)
    print("----------------------------------------\n")
```

```
Q: When did Tesla become a vegetarian?
A: his later years
----------------------------------------

Q: When did Tesla move to United States ?
A: 1884
----------------------------------------

Q: What year did Tesla die?
A: 1943
----------------------------------------

Q: Who edited the book My Inventions: The Autobiography of Nikola Tesla?
A: Ben Johnston
----------------------------------------

Q: In what age did Tesla died?
A: 86
```

```
----------------------------------------

Q: Who developed an induction motor?
A: Tesla
----------------------------------------

Q: Where Tesla was born?
A: Austrian Empire
----------------------------------------

Q: What did Tesla study?
A: engineering and physics
----------------------------------------
```

References

- https://github.com/nlpyang/pytorch-transformers/tree/master/examples
- https://github.com/flogothetis/SQuAD-QueryAnswering-BERT-Keras/blob/main/SQuAD_QuestionAnswering_Bert.ipynb
- https://github.com/dredwardhyde/bert-examples/blob/main/bert_squad_pytorch.py?fbclid=IwAR1VGhZx6MsVlOha3lDX_uC8PASSDu9ECKceD2XCHGSetKhldgay0F8SirY
- https://colab.research.google.com/drive/1Zp2_Uka8oGDYsSe5ELk-xz6wIX8OlkB7?fbclid=IwAR1zl-nOBSOdYA4H-WY-ba6AJ--hRHM2OZhgK3DrQ1SLfavln5M-k-r4jJ4#scrollTo=j3_CAQUf2asD