```
#pip install transformers

#load all libraries
import requests
import json
import torch
import torch.nn as nn
import os
from tqdm import tqdm
from transformers import BertModel, BertTokenizerFast, AdamW
# AutoTokenizer, AutoModelForQuestionAnswering, BertTokenizer,
BertForQuestionAnswering
from torch.utils.data import Dataset, DataLoader
from torch.optim.lr_scheduler import ExponentialLR
import matplotlib.pyplot as plt

MODEL_PATH = "bert-base-uncased"
```

Load Dataset

```
#get SQuAD v2
!wget -nc https://rajpurkar.github.io/SQuAD-explorer/dataset/train-
v2.0.json
!wget -nc https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-
v2.0.json

--2024-11-14 07:23:41--
https://rajpurkar.github.io/SQuAD-explorer/dataset/train-v2.0.json
Resolving rajpurkar.github.io (rajpurkar.github.io)...
185.199.108.153, 185.199.109.153, 185.199.110.153, ...
Connecting to rajpurkar.github.io (rajpurkar.github.io)|
185.199.108.153|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 42123633 (40M) [application/json]
Saving to: ‘train-v2.0.json’

train-v2.0.json     100%[===================>]  40.17M  45.3MB/s    in
0.9s

2024-11-14 07:23:42 (45.3 MB/s) - ‘train-v2.0.json’ saved
[42123633/42123633]

--2024-11-14 07:23:42--
https://rajpurkar.github.io/SQuAD-explorer/dataset/dev-v2.0.json
Resolving rajpurkar.github.io (rajpurkar.github.io)...
185.199.108.153, 185.199.109.153, 185.199.110.153, ...
Connecting to rajpurkar.github.io (rajpurkar.github.io)|
185.199.108.153|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 4370528 (4.2M) [application/json]
```

```
Saving to: ‘dev-v2.0.json’

dev-v2.0.json       100%[===================>]   4.17M  11.6MB/s    in
0.4s

2024-11-14 07:23:43 (11.6 MB/s) - ‘dev-v2.0.json’ saved
[4370528/4370528]
```

```python
num_questions = 0
num_posible = 0
num_imposible = 0

#note: below code will only return questions wich have answers (i.e.
not the ones flagged as imposible to answer)
def get_data(path):
    #read each file and retrieve the contexts, qustions and answers
  with open(path, 'rb') as f:
    raw_data = json.load(f)

  contexts = []
  questions = []
  answers = []
  num_q = 0
  num_pos = 0
  num_imp = 0

  for group in raw_data['data']:
    for paragraph in group['paragraphs']:
      context = paragraph['context']
      for qa in paragraph['qas']:
        question = qa['question']
        num_q  = num_q  +1
        if qa['is_impossible'] == True:
            num_imp = num_imp +1
        else:
            num_pos = num_pos +1
        for answer in qa['answers']:
          contexts.append(context.lower())
          questions.append(question.lower())
          answers.append(answer)

  return num_q, num_pos, num_imp, contexts, questions, answers

num_q, num_pos, num_imp, train_contexts, train_questions,
train_answers = get_data('train-v2.0.json')
num_questions  = num_q
num_posible = num_pos
num_imposible  = num_imp
```

```
print(train_questions[0:10])
print(train_answers[0:10])
```

```
['when did beyonce start becoming popular?', 'what areas did beyonce
compete in when she was growing up?', "when did beyonce leave
destiny's child and become a solo singer?", 'in what city and state
did beyonce  grow up? ', 'in which decade did beyonce become famous?',
'in what r&b group was she the lead singer?', 'what album made her a
worldwide known artist?', "who managed the destiny's child group?",
'when did beyoncé rise to fame?', "what role did beyoncé have in
destiny's child?"]
[{'text': 'in the late 1990s', 'answer_start': 269}, {'text': 'singing
and dancing', 'answer_start': 207}, {'text': '2003', 'answer_start':
526}, {'text': 'Houston, Texas', 'answer_start': 166}, {'text': 'late
1990s', 'answer_start': 276}, {'text': "Destiny's Child",
'answer_start': 320}, {'text': 'Dangerously in Love', 'answer_start':
505}, {'text': 'Mathew Knowles', 'answer_start': 360}, {'text': 'late
1990s', 'answer_start': 276}, {'text': 'lead singer', 'answer_start':
290}]
```

```
num_q, num_pos, num_imp, valid_contexts, valid_questions,
valid_answers = get_data('dev-v2.0.json')
num_questions  = num_questions + num_q
num_posible = num_posible + num_pos
num_imposible = num_imposible  + num_imp

print(f"Total number of questions: {num_questions}")
print(f"Total number of Answerable questions: {num_posible}")
print(f"Total number of impossible questions: {num_imposible}")
```

```
Total number of questions: 142192
Total number of Answerable questions: 92749
Total number of impossible questions: 49443
```

```
print(valid_questions[0:10])
print(valid_answers[0:10])
```

```
['in what country is normandy located?', 'in what country is normandy
located?', 'in what country is normandy located?', 'in what country is
normandy located?', 'when were the normans in normandy?', 'when were
the normans in normandy?', 'when were the normans in normandy?', 'when
were the normans in normandy?', 'from which countries did the norse
originate?', 'from which countries did the norse originate?']
[{'text': 'France', 'answer_start': 159}, {'text': 'France',
'answer_start': 159}, {'text': 'France', 'answer_start': 159},
{'text': 'France', 'answer_start': 159}, {'text': '10th and 11th
centuries', 'answer_start': 94}, {'text': 'in the 10th and 11th
centuries', 'answer_start': 87}, {'text': '10th and 11th centuries',
'answer_start': 94}, {'text': '10th and 11th centuries',
'answer_start': 94}, {'text': 'Denmark, Iceland and Norway',
```

```
'answer_start': 256}, {'text': 'Denmark, Iceland and Norway',
'answer_start': 256}]

def add_answer_end(answers, contexts):
  for answer, context in zip(answers, contexts):
    answer['text'] = answer['text'].lower()
    answer['answer_end'] = answer['answer_start'] +
len(answer['text'])

add_answer_end(train_answers, train_contexts)
add_answer_end(valid_answers, valid_contexts)

print(f"Context: {train_contexts[0]}")
print(f"Question: {train_questions[0]}")
print(f"Answer: {train_answers[0]}")

Context: beyoncé giselle knowles-carter (/biːˈjɒnseɪ/ bee-yon-say)
(born september 4, 1981) is an american singer, songwriter, record
producer and actress. born and raised in houston, texas, she performed
in various singing and dancing competitions as a child, and rose to
fame in the late 1990s as lead singer of r&b girl-group destiny's
child. managed by her father, mathew knowles, the group became one of
the world's best-selling girl groups of all time. their hiatus saw the
release of beyoncé's debut album, dangerously in love (2003), which
established her as a solo artist worldwide, earned five grammy awards
and featured the billboard hot 100 number-one singles "crazy in love"
and "baby boy".
Question: when did beyonce start becoming popular?
Answer: {'text': 'in the late 1990s', 'answer_start': 269,
'answer_end': 286}

test_rec = 30
print(f"Context: {valid_contexts[test_rec]}")
print(f"Question: {valid_questions[test_rec]}")
print(f"Answer: {valid_answers[test_rec]}")

Context: the english name "normans" comes from the french words
normans/normanz, plural of normant, modern french normand, which is
itself borrowed from old low franconian nortmann "northman" or
directly from old norse norðmaðr, latinized variously as nortmannus,
normannus, or nordmannus (recorded in medieval latin, 9th century) to
mean "norseman, viking".
Question: what is the original meaning of the word norman?
Answer: {'text': 'norseman, viking', 'answer_start': 331,
'answer_end': 347}
```

Tokenize

Find max lengths

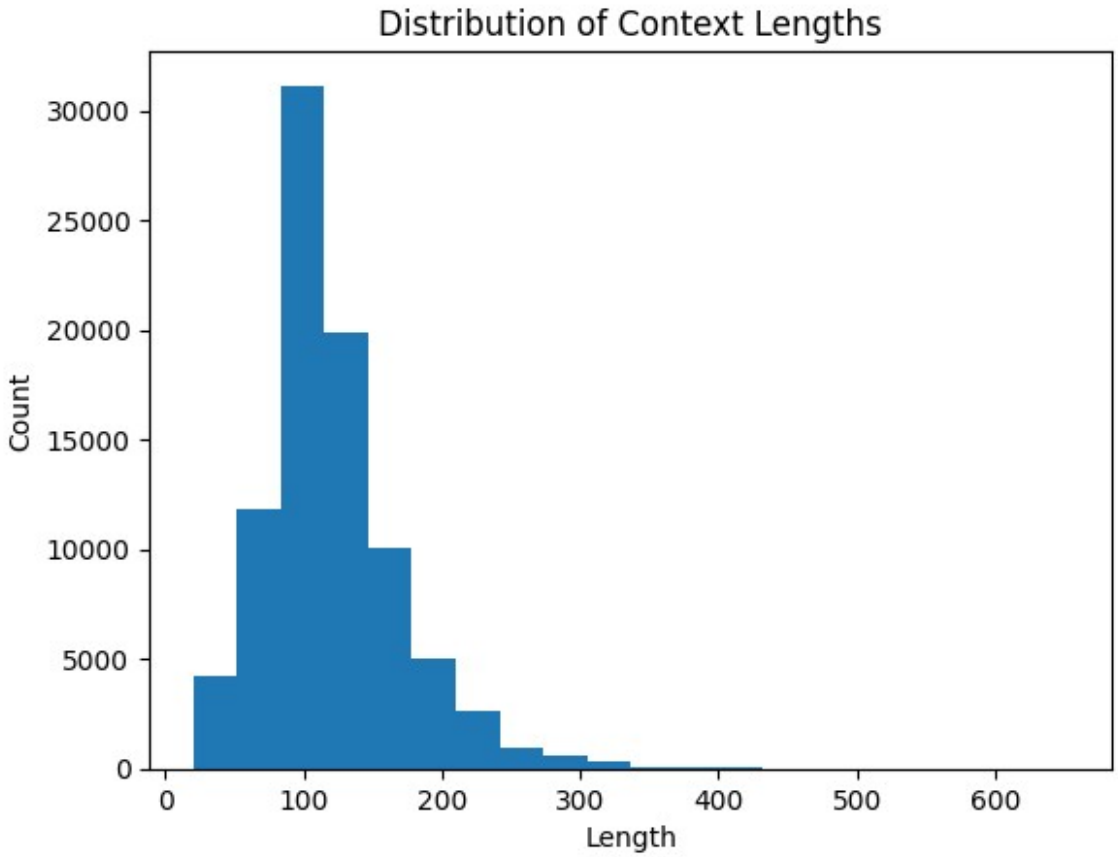
```
#Text lengths to contextx
token_lens = []

for txt in train_contexts:
    txt = txt.strip()  # remove leading and trailing whitespaces
    token_lens.append(len(txt.split(' ')))


print(max(token_lens))

plt.hist(token_lens,  bins=20)  # density=False would make counts
plt.ylabel('Count')
plt.xlabel('Length')
plt.title('Distribution of Context Lengths');
```

```
653
```



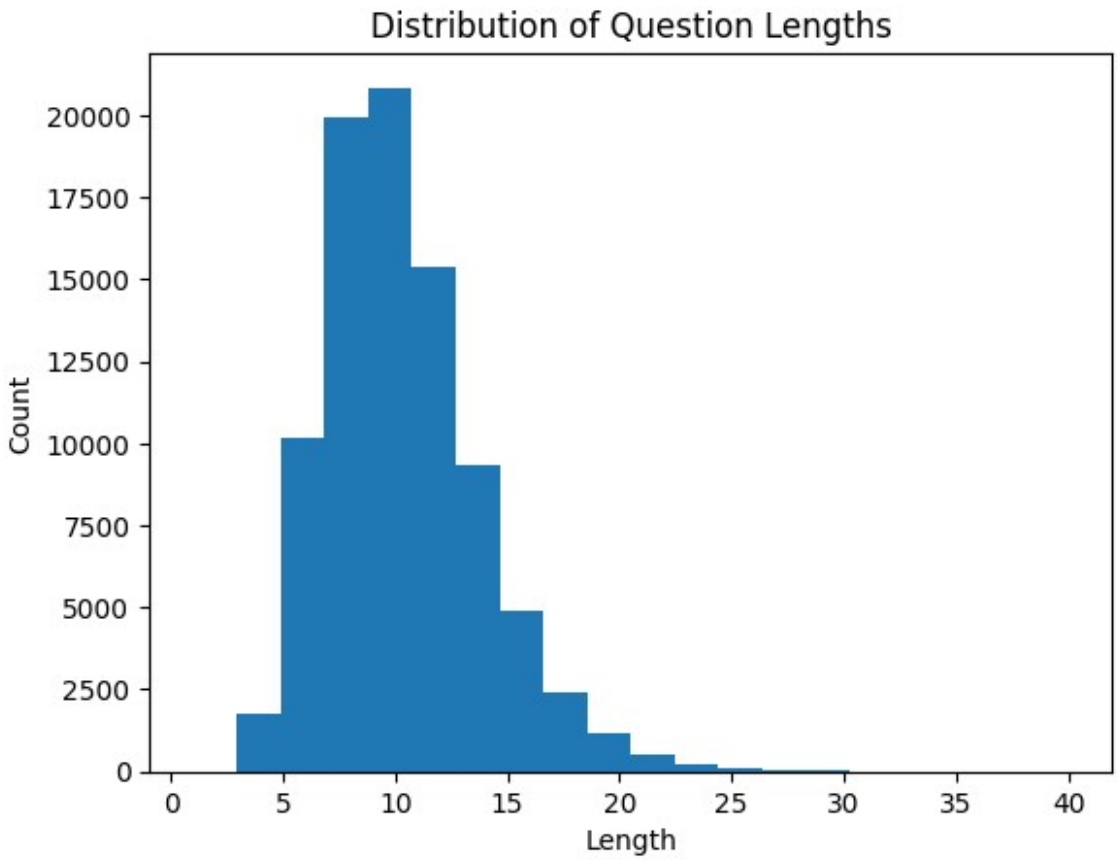

```
#Test lengths of Questions
token_lens2 = []
```

```
for txt in train_questions:
    txt = txt.strip()  # remove leading and trailing whitespaces
    token_lens2.append(len(txt.split(' ')))


print(max(token_lens2))
print(len(token_lens2))

plt.hist(token_lens2,  bins=20)  # density=False would make counts
plt.ylabel('Count')
plt.xlabel('Length')
plt.title('Distribution of Question Lengths');
```

```
40
86821
```




```
MAX_LENGTH = 250
```

tokenize

```
tokenizerFast = BertTokenizerFast.from_pretrained(MODEL_PATH)
```

```
train_encodings_fast = tokenizerFast(train_questions, train_contexts,
max_length = MAX_LENGTH, truncation=True, padding=True)
valid_encodings_fast = tokenizerFast(valid_questions,valid_contexts,
max_length = MAX_LENGTH, truncation=True, padding=True)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/
_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your
settings tab (https://huggingface.co/settings/tokens), set it as
secret in your Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to
access public models or datasets.
  warnings.warn(
```


```
{"model_id":"c3ecb11771314edd95a04a4cf08058ed","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"6042f95326034f81bb782b8eecdbc92b","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"eb600098b8344e9fb4edc8eeccdb4210","version_major":2,"vers
ion_minor":0}
```

```
{"model_id":"93a1439a34e84cf09c6ad395f39548f9","version_major":2,"vers
ion_minor":0}
```


```
type(train_encodings_fast)
```

```
transformers.tokenization_utils_base.BatchEncoding
```

```
print(train_encodings_fast.keys())
print(valid_encodings_fast.keys())
print(len(train_encodings_fast['input_ids']))
print(len(train_encodings_fast['input_ids'][0]))
```

```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
86821
250
```

```
print(train_encodings_fast['input_ids'][0])
```

```
[101, 2043, 2106, 20773, 2707, 3352, 2759, 1029, 102, 20773, 21025,
19358, 22815, 1011, 5708, 1006, 1013, 12170, 23432, 29715, 3501,
29678, 12325, 29685, 1013, 10506, 1011, 10930, 2078, 1011, 2360, 1007,
1006, 2141, 2244, 1018, 1010, 3261, 1007, 2003, 2019, 2137, 3220,
1010, 6009, 1010, 2501, 3135, 1998, 3883, 1012, 2141, 1998, 2992,
1999, 5395, 1010, 3146, 1010, 2016, 2864, 1999, 2536, 4823, 1998,
5613, 6479, 2004, 1037, 2775, 1010, 1998, 3123, 2000, 4476, 1999,
```

```
1996, 2397, 4134, 2004, 2599, 3220, 1997, 1054, 1004, 1038, 2611,
1011, 2177, 10461, 1005, 1055, 2775, 1012, 3266, 2011, 2014, 2269,
1010, 25436, 22815, 1010, 1996, 2177, 2150, 2028, 1997, 1996, 2088,
1005, 1055, 2190, 1011, 4855, 2611, 2967, 1997, 2035, 2051, 1012,
2037, 14221, 2387, 1996, 2713, 1997, 20773, 1005, 1055, 2834, 2201,
1010, 20754, 1999, 2293, 1006, 2494, 1007, 1010, 2029, 2511, 2014,
2004, 1037, 3948, 3063, 4969, 1010, 3687, 2274, 8922, 2982, 1998,
2956, 1996, 4908, 2980, 2531, 2193, 1011, 2028, 3895, 1000, 4689,
1999, 2293, 1000, 1998, 1000, 3336, 2879, 1000, 1012, 102, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0]

type(train_answers[0]['text'])
train_answers[0].keys()

dict_keys(['text', 'answer_start', 'answer_end'])

#train_answer_encodings_fast = tokenizerFast(train_answers[0]['text'],
max_length = MAX_LENGTH, truncation=True, padding=True)

def ret_Answer_start_and_end_train(idx):
    ret_start = 0
    ret_end = 0
    answer_encoding_fast = tokenizerFast(train_answers[idx]['text'],
max_length = MAX_LENGTH, truncation=True, padding=True)
    for a in range( len(train_encodings_fast['input_ids'][idx]) -
len(answer_encoding_fast['input_ids']) ):
#len(train_encodings_fast['input_ids'][0])):
        match = True
        for i in range(1,len(answer_encoding_fast['input_ids']) - 1):

            if (answer_encoding_fast['input_ids'][i] !=
train_encodings_fast['input_ids'][idx][a + i]):
                match = False
                break
        if match:
            ret_start = a+1
            ret_end = a+i+1
            break
    return(ret_start, ret_end)

test_rec=92

z,x = ret_Answer_start_and_end_train(test_rec)
print(z, x)

predict_answer_tokens = train_encodings_fast.input_ids[test_rec][z :
x]
print(tokenizerFast.decode(predict_answer_tokens))
```

```python
print(train_answers[test_rec]['text'])
print(tokenizerFast.decode(train_encodings_fast['input_ids']
[test_rec]))
```

```
0 0

split with luckett and rober
[CLS] what event caused beyonce ' s depression? [SEP] letoya luckett
and roberson became unhappy with mathew ' s managing of the band and
eventually were replaced by farrah franklin and michelle williams.
beyonce experienced depression following the split with luckett and
roberson after being publicly blamed by the media, critics, and blogs
for its cause. her long - standing boyfriend left her at this time.
the depression was so severe it lasted for a couple of years, during
which she occasionally kept herself in her bedroom for days and
refused to eat anything. beyonce stated that she struggled to speak
about her depression because destiny ' s child had just won their
first grammy award and she feared no one would take her seriously.
beyonce would later speak of her mother as the person who helped her
fight it. franklin was dismissed, leaving just beyonce, rowland, and
williams. [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] [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]
```

```python
print(train_encodings_fast.keys())
print(valid_encodings_fast.keys())
print(len(train_encodings_fast['input_ids']))
```

```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
86821
```

```python
start_positions = []
end_positions = []
ctr = 0
for h in range(len(train_encodings_fast['input_ids'])):
    #print(h)
    s, e = ret_Answer_start_and_end_train(h)
    start_positions.append(s)
    end_positions.append(e)
    if s==0:
        ctr = ctr + 1


train_encodings_fast.update({'start_positions': start_positions,
```

```
'end_positions': end_positions})
print(ctr)
```

```
1190
```

```
print(train_encodings_fast.keys())
print(valid_encodings_fast.keys())
print(len(train_encodings_fast['input_ids']))
```

```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask',
'start_positions', 'end_positions'])
dict_keys(['input_ids', 'token_type_ids', 'attention_mask'])
86821
```

```
test_rec = 1
print(train_encodings_fast['start_positions'][test_rec])
print(train_encodings_fast['end_positions'][test_rec])
predict_answer_tokens = train_encodings_fast.input_ids[test_rec]
[train_encodings_fast['start_positions'][test_rec] :
train_encodings_fast['end_positions'][test_rec]]
print(tokenizerFast.decode(predict_answer_tokens))
print(train_answers[test_rec]['text'])
print(tokenizerFast.decode(train_encodings_fast['input_ids']
[test_rec]))
```

```
68
71
singing and dancing
singing and dancing
[CLS] what areas did beyonce compete in when she was growing up? [SEP]
beyonce giselle knowles - carter ( / biːˈjɒnseɪ / bee - yon - say )
( born september 4, 1981 ) is an american singer, songwriter, record
producer and actress. born and raised in houston, texas, she performed
in various singing and dancing competitions as a child, and rose to
fame in the late 1990s as lead singer of r & b girl - group destiny '
s child. managed by her father, mathew knowles, the group became one
of the world ' s best - selling girl groups of all time. their hiatus
saw the release of beyonce ' s debut album, dangerously in love ( 2003
), which established her as a solo artist worldwide, earned five
grammy awards and featured the billboard hot 100 number - one singles
" crazy in love " and " baby boy ". [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] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD]
```

```
def ret_Answer_start_and_end_valid(idx):
    ret_start = 0
```

```
    ret_end = 0
    answer_encoding_fast = tokenizerFast(valid_answers[idx]['text'],
max_length = MAX_LENGTH, truncation=True, padding=True)
    for a in range( len(valid_encodings_fast['input_ids'][idx])  -
len(answer_encoding_fast['input_ids'])   ):
#len(train_encodings_fast['input_ids'][0])):
        match = True
        for i in range(1,len(answer_encoding_fast['input_ids']) - 1):
            if (answer_encoding_fast['input_ids'][i] !=
valid_encodings_fast['input_ids'][idx][a + i]):
                match = False
                break
        if match:
            ret_start = a+1
            ret_end = a+i+1
            break
    return(ret_start, ret_end)

start_positions = []
end_positions = []
ctr = 0
for h in range(len(valid_encodings_fast['input_ids']) ):
    #print(h)
    s, e = ret_Answer_start_and_end_valid(h)
    start_positions.append(s)
    end_positions.append(e)
    if s==0:
        ctr = ctr + 1


valid_encodings_fast.update({'start_positions': start_positions,
'end_positions': end_positions})
print(ctr)
```

```
393
```

```
test_rec=2

z,x = ret_Answer_start_and_end_valid(test_rec)

predict_answer_tokens = valid_encodings_fast.input_ids[test_rec][z :
x]
print(tokenizerFast.decode(predict_answer_tokens))
print(valid_answers[test_rec]['text'])
print(tokenizerFast.decode(valid_encodings_fast['input_ids']
[test_rec]))
```

```
france
france
[CLS] in what country is normandy located? [SEP] the normans
( norman : nourmands ; french : normands ; latin : normanni ) were the
```

```
people who in the 10th and 11th centuries gave their name to normandy,
a region in france. they were descended from norse ( " norman " comes
from " norseman " ) raiders and pirates from denmark, iceland and
norway who, under their leader rollo, agreed to swear fealty to king
charles iii of west francia. through generations of assimilation and
mixing with the native frankish and roman - gaulish populations, their
descendants would gradually merge with the carolingian - based
cultures of west francia. the distinct cultural and ethnic identity of
the normans emerged initially in the first half of the 10th century,
and it continued to evolve over the succeeding centuries. [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] [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]
```

```python
print(train_encodings_fast.keys())
print(valid_encodings_fast.keys())
print(len(train_encodings_fast['input_ids']))
print(len(train_encodings_fast['start_positions']))
print(len(train_encodings_fast['end_positions']))
print(len(valid_encodings_fast['input_ids']))
print(len(valid_encodings_fast['start_positions']))
print(len(valid_encodings_fast['end_positions']))
```

```
dict_keys(['input_ids', 'token_type_ids', 'attention_mask',
'start_positions', 'end_positions'])
dict_keys(['input_ids', 'token_type_ids', 'attention_mask',
'start_positions', 'end_positions'])
86821
86821
86821
20302
20302
20302
```

```python
#tokenizerFast.decode(train_encodings_fast['input_ids'][0])
```

```python
tokenizerFast.decode(train_encodings_fast['input_ids'][0])
```

{"type":"string"}

```python
test_row= 0
print(train_contexts[test_row][train_answers[test_row]
['answer_start']:train_answers[test_row]['answer_end']])
```

```
in the late 1990s
```

```
answer_start_index = 75
answer_end_index = 79

predict_answer_tokens = train_encodings_fast.input_ids[0]
[answer_start_index : answer_end_index]
tokenizerFast.decode(predict_answer_tokens)
```

```
{"type":"string"}
```

OPTIONAL: try a pretrained model. (uncomment to test out)

```
#tokenizer2 = BertTokenizer.from_pretrained("deepset/bert-base-cased-
squad2")
#model2 = BertForQuestionAnswering.from_pretrained("deepset/bert-base-
cased-squad2")

#question= train_questions[0]
#text = train_contexts[0]

#print(question)
#print(text)

#inputs = tokenizer2(question, text, return_tensors="pt")
#with torch.no_grad():
#    outputs = model2(**inputs)

#answer_start_index = outputs.start_logits.argmax()
#answer_end_index = outputs.end_logits.argmax()
#print(answer_start_index)
#print(answer_end_index)


#predict_answer_tokens = inputs.input_ids[0, answer_start_index :
answer_end_index + 1]
#tokenizer2.decode(predict_answer_tokens)
```

Create Dataset and Dataloaders

```
class InputDataset(Dataset):
    def __init__(self, encodings):
        self.encodings = encodings
    def __getitem__(self, i):
        return {
            'input_ids': torch.tensor(self.encodings['input_ids'][i]),
            'token_type_ids':
torch.tensor(self.encodings['token_type_ids'][i]),
            'attention_mask':
```

```
torch.tensor(self.encodings['attention_mask'][i]),
            'start_positions':
torch.tensor(self.encodings['start_positions'][i]),
            'end_positions':
torch.tensor(self.encodings['end_positions'][i])
        }
    def __len__(self):
        return len(self.encodings['input_ids'])

train_dataset = InputDataset(train_encodings_fast)
valid_dataset = InputDataset(valid_encodings_fast)

print(len(train_dataset))
print(train_dataset[0].keys())
```

```
86821
dict_keys(['input_ids', 'token_type_ids', 'attention_mask',
'start_positions', 'end_positions'])
```

```
train_data_loader = DataLoader(train_dataset, batch_size=16,
shuffle=True)
valid_data_loader = DataLoader(valid_dataset, batch_size=32)


#print(data['targets'].shape)
```

Create Model

```
#model = BertForQuestionAnswering.from_pretrained(MODEL_PATH)

bert_model = BertModel.from_pretrained(MODEL_PATH)  #MODEL_PATH =
"bert-base-uncased"

class QAModel(nn.Module):
    def __init__(self):
        super(QAModel, self).__init__()
        self.bert = bert_model
        self.drop_out = nn.Dropout(0.1)
        self.l1 = nn.Linear(768 * 2, 768 * 2)
        self.l2 = nn.Linear(768 * 2, 2)
        self.linear_relu_stack = nn.Sequential(
            self.drop_out,
            self.l1,
            nn.LeakyReLU(),
            self.l2
        )

    def forward(self, input_ids, attention_mask, token_type_ids):
        model_output = self.bert(input_ids,
attention_mask=attention_mask, token_type_ids=token_type_ids,
output_hidden_states=True)
```

```
        hidden_states = model_output[2]
        out = torch.cat((hidden_states[-1], hidden_states[-3]), dim=-
1)  # taking Start logits from last BERT layer, End Logits from third
to last layer
        logits = self.linear_relu_stack(out)

        start_logits, end_logits = logits.split(1, dim=-1)

        start_logits = start_logits.squeeze(-1)
        end_logits = end_logits.squeeze(-1)

        return start_logits, end_logits
```


```
{"model_id":"e237fcfa4c5c48c69907f87a96c7071d","version_major":2,"vers
ion_minor":0}
```


```
model = QAModel()

device = torch.device('cuda') if torch.cuda.is_available() else
torch.device('cpu')
print(device)
```

```
cuda
```

Create Loss Functions

```
# my function to manually calculate Cross Entropy Loss
def loss_fn(start_logits, end_logits, start_positions, end_positions):
    loss_fct = nn.CrossEntropyLoss()
    start_loss = loss_fct(start_logits, start_positions)
    end_loss = loss_fct(end_logits, end_positions)
    total_loss = (start_loss + end_loss)/2
    return total_loss

# my focal loss function. Focal Loss = (True Vector)*((1 -
probs)^Gamma)*log(probs)
# where Gamma is a factor we use. setting Gamma = 0 makes this a Cross
Entropy Loss function

def focal_loss_fn(start_logits, end_logits, start_positions,
end_positions, gamma):

    #calculate Probabilities by applying Softmax to the Start and End
Logits. Then get 1 - probabilities
    smax = nn.Softmax(dim=1)
    probs_start = smax(start_logits)
    inv_probs_start = 1 - probs_start
    probs_end = smax(end_logits)
    inv_probs_end = 1 - probs_end

    #get log of probabilities. Note: NLLLoss required log
```

```
probabilities. This is the Natural Log (Log base e)
    lsmax = nn.LogSoftmax(dim=1)
    log_probs_start = lsmax(start_logits)
    log_probs_end = lsmax(end_logits)

    nll = nn.NLLLoss()

    fl_start = nll(torch.pow(inv_probs_start, gamma)* log_probs_start,
start_positions)
    fl_end = nll(torch.pow(inv_probs_end, gamma)*log_probs_end,
end_positions)

    #return mean of the Loss for the start and end logits
    return ((fl_start + fl_end)/2)
```

OPTIONAL: Uncomment this code if you want to test on one input .....

```
#data = next(iter(train_data_loader))
#data.keys()

#print(data['input_ids'].shape)
#print(data['attention_mask'].shape)

#run one row
#model.to(device)
#model.train()
#input_ids = data['input_ids'][0].unsqueeze(0).to(device)
#attention_mask = data['attention_mask'][0].unsqueeze(0).to(device)
#start_positions = data['start_positions'][0].unsqueeze(0).to(device)
#end_positions = data['end_positions'][0].unsqueeze(0).to(device)

#out_start, out_end = model(input_ids=input_ids,
attention_mask=attention_mask, token_type_ids=token_type_ids)

#print(f"start logits shape: {out_start.shape}")
#print(f"end logits shape: {out_end.shape}")

#answer_start_index = out_start.argmax()
#answer_end_index = out_end.argmax()
#print(answer_start_index)
#print(answer_end_index)
```

Train Loop

```
optim = AdamW(model.parameters(), lr=2e-5, weight_decay=2e-2)
scheduler = ExponentialLR(optim, gamma=0.9)
total_acc = []
total_loss = []
```

```
/usr/local/lib/python3.10/dist-packages/transformers/
optimization.py:591: FutureWarning: This implementation of AdamW is
```

```
deprecated and will be removed in a future version. Use the PyTorch
implementation torch.optim.AdamW instead, or set
`no_deprecation_warning=True` to disable this warning
  warnings.warn(
```

```
def train_epoch(model, dataloader, epoch):
    model = model.train()
    losses = []
    acc = []
    ctr = 0
    batch_tracker = 0
    for batch in tqdm(dataloader, desc = 'Running Epoch '):
        optim.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        token_type_ids = batch['token_type_ids'].to(device)
        start_positions = batch['start_positions'].to(device)
        end_positions = batch['end_positions'].to(device)
        out_start, out_end = model(input_ids=input_ids,
                attention_mask=attention_mask,
                token_type_ids=token_type_ids)
        #loss = loss_fn(out_start, out_end, start_positions,
end_positions)  # <---BASELINE.  Cross Entropy Loss is returned by
Default
        loss = focal_loss_fn(out_start, out_end, start_positions,
end_positions,1) #using gamma = 1
        losses.append(loss.item())
        loss.backward()
        optim.step()

        start_pred = torch.argmax(out_start, dim=1)
        end_pred = torch.argmax(out_end, dim=1)

        acc.append(((start_pred ==
start_positions).sum()/len(start_pred)).item())
        acc.append(((end_pred ==
end_positions).sum()/len(end_pred)).item())
        #ctr = ctr +1
        #if ctr==50:
        #    break
        batch_tracker = batch_tracker + 1
        if batch_tracker==250 and epoch==1:
            total_acc.append(sum(acc)/len(acc))
            loss_avg = sum(losses)/len(losses)
            total_loss.append(loss_avg)
            batch_tracker = 0
    scheduler.step()
    ret_acc = sum(acc)/len(acc)
    ret_loss = sum(losses)/len(losses)
    return(ret_acc, ret_loss)
```

```
def eval_model(model, dataloader):
    model = model.eval()
    losses = []
    acc = []
    ctr = 0
    with torch.no_grad():
        for batch in tqdm(dataloader, desc = 'Running Evaluation'):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            token_type_ids = batch['token_type_ids'].to(device)
            start_true = batch['start_positions'].to(device)
            end_true = batch['end_positions'].to(device)

            out_start, out_end = model(input_ids=input_ids,
                attention_mask=attention_mask,
                token_type_ids=token_type_ids)

            start_pred = torch.argmax(out_start, dim=1)
            end_pred = torch.argmax(out_end, dim=1)

            acc.append(((start_pred ==
start_true).sum()/len(start_pred)).item())
            acc.append(((end_pred ==
end_true).sum()/len(end_pred)).item())
            #ctr = ctr +1
            #if ctr==50:
            #    break
        ret_acc = sum(acc)/len(acc)
        ret_loss = 0
        #ret_loss = sum(losses)/len(losses)
    return(ret_acc)

EPOCHS = 4

model.to(device)

for epoch in range(EPOCHS):
    train_acc, train_loss = train_epoch(model, train_data_loader,
epoch+1)
    print(f"Train Accuracy: {train_acc}      Train Loss:
{train_loss}")
    val_acc = eval_model(model, valid_data_loader)
    print(f"Validation Accuracy: {val_acc}")

#val_acc, val_loss = eval_model(model, valid_data_loader)
#print(f"Validation Accuracy: {val_acc}   Validation Loss:
{val_loss}")

Running Epoch :   5%|▌          | 260/5427 [03:03<1:01:52,  1.39it/s]

scheduler.get_last_lr()
```

```
# plot Accuracy
plt.plot(total_acc, color='blue')
plt.ylabel('Accuracy')
plt.xlabel('Runs->')
plt.title("Total Train Accuracy over time");

# plot Loss
plt.plot(total_loss, color='red')
plt.ylabel('Loss')
plt.xlabel('Runs->')
plt.title("Total Train Loss over time");

torch.save(model.state_dict(), '/kaggle/working/best_model_state.bin')
```

Test

```
def get_answer(question, context):
    inputs = tokenizerFast.encode_plus(question, context,
return_tensors='pt').to(device)
    with torch.no_grad():
        output_start, output_end = model(**inputs)

        answer_start = torch.argmax(output_start)
        answer_end = torch.argmax(output_end)

        answer =
tokenizerFast.convert_tokens_to_string(tokenizerFast.convert_ids_to_to
kens(inputs['input_ids'][0][answer_start:answer_end]))

        return(answer)

test_rec = 0

print(f"Context: {valid_contexts[test_rec]}")
print(f"Question: {valid_questions[test_rec]}")
print(f"Expected Answer: {valid_answers[test_rec]}")

context = valid_contexts[test_rec]
question = valid_questions[test_rec]

print(f"Predicted Answer: {get_answer(question, context)}")

test_rec = 15

print(f"Context: {valid_contexts[test_rec]}")
print(f"Question: {valid_questions[test_rec]}")
print(f"Expected Answer: {valid_answers[test_rec]}")

context = valid_contexts[test_rec]
question = valid_questions[test_rec]
```

```
print(f"Predicted Answer: {get_answer(question, context)}")

test_rec = 28

print(f"Context: {valid_contexts[test_rec]}")
print(f"Question: {valid_questions[test_rec]}")
print(f"Expected Answer: {valid_answers[test_rec]}")

context = valid_contexts[test_rec]
question = valid_questions[test_rec]

print(f"Predicted Answer: {get_answer(question, context)}")

test_rec = 2000

print(f"Context: {valid_contexts[test_rec]}")
print(f"Question: {valid_questions[test_rec]}")
print(f"Expected Answer: {valid_answers[test_rec]}")

context = valid_contexts[test_rec]
question = valid_questions[test_rec]

print(f"Predicted Answer: {get_answer(question, context)}")

test_rec = 4000

print(f"Context: {valid_contexts[test_rec]}")
print(f"Question: {valid_questions[test_rec]}")
print(f"Expected Answer: {valid_answers[test_rec]}")

context = valid_contexts[test_rec]
question = valid_questions[test_rec]

print(f"Predicted Answer: {get_answer(question, context)}")
```