## DATA COLLECTION AND PROCESSING

```
from google.colab import drive
drive.mount('/content/drive')
```

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

```
import requests
from bs4 import BeautifulSoup
from tqdm import tqdm
import re
import pandas as pd

def clean_text(text):

    text = re.sub(r'^(Q\.|[A-Z\s]+:)', '', text.strip())

    text = re.sub(r'[^\w\s.?!\'"]', '', text)

    return text

def scrape_interview(interview_id):
    url = f"https://www.asapsports.com/show_interview.php?id={interview_id}"

    response = requests.get(url)
    if response.status_code == 200:
        soup = BeautifulSoup(response.content, 'html.parser')

        q = []
        a = []

        #print(soup.get_text())

        lines = soup.get_text().split('\n')
        question = False
        for line in lines:
          if 'Q.' in line:
            q.append(clean_text(line))
            question = True
          elif question == True and (':' in line):
            a.append(clean_text(line))
            question = False

        min_length = min(len(q), len(a))
        a = a[:min_length]
        q = q[:min_length]
        return q, a

    else:
        print(f"Failed ID: {interview_id}")
        return [], []

interview_ids = range(160001, 193174)  # Example list of interview IDs

all_questions = []
all_answers = []
id_str = 0

for interview_id in tqdm(interview_ids):

    if interview_id % 10000 == 0:
      dict = {'questions': all_questions, 'answers': all_answers}

      df = pd.DataFrame(dict)

      id_str += 1
      #try:
      df.to_csv('./QAData' + str(id_str) + '.csv', index=False)
      #except:
      #  df.to_csv('/content/drive/QAData' + str(id_str) + '.csv', index=False)


    questions, answers = scrape_interview(interview_id)
    all_questions.extend(questions)
    all_answers.extend(answers)

#for i in range(len(all_questions)):
#    print(f"Question {i + 1}: {all_questions[i]}")
#    print(f"Answer {i + 1}: {all_answers[i]}")
#    print()

dict = {'questions': all_questions, 'answers': all_answers}

df = pd.DataFrame(dict)

try:
  df.to_csv('/content/drive/My Drive/QAData.csv', index=False)
except:
  df.to_csv('/content/drive/QAData.csv', index=False)
```

```
 46%|█████     | 15408/33173 [1:42:42<1:36:27,  3.07it/s]WARNING:bs4.dammit:S
100%|██████████| 33173/33173 [3:40:56<00:00,  2.50it/s]
```

```
#final_0_1 = pd.read_csv('./QAData0_1.csv')
```

```
#final_0_2 = pd.read_csv('./QAData0_2.csv')
#final = pd.concat([final_0_1, final_0_2],ignore_index=True)

df1 = pd.read_csv('./QAData1.csv')
df2 = pd.read_csv('./QAData2.csv')
df3 = pd.read_csv('./QAData3.csv')
df4 = pd.read_csv('./QAData4.csv')
df5 = pd.read_csv('./QAData5.csv')
df6 = pd.read_csv('./QAData6.csv')
df7 = pd.read_csv('./QAData7.csv')

final = pd.concat([df1,df2,df3,df4,df5,df6,df7])

all_questions = final['questions'].tolist()
all_answers = final['answers'].tolist()

print(len(all_questions))
print(len(all_answers))
for i in range(5):
  print("Q:", all_questions[i])
  print("A:", all_answers[i])
  print()
```

```
344072
344072
Q:  You're back here in this place where you've won before.  Indiscernible  w
A:  I love the golf course but the fans are what makes it awesome.  To be abl

Q:  I caught a glimpse of you looking at the wall of achievements.  I can't i
A:  Yeah it's an honor.  A lot of names up there that I would've loved to pla

Q:  In terms of making your comeback now can you give us an update on how you
A:  I'm just taking it one day at a time really.  It's been a struggle.  I'm

Q:  How tough is it to fight through that when you're trying to find the cons
A:  Yeah it's pretty tough.  It's pretty tough to find.  I don't know man.  I

Q:  Does it give you a different perspective having to fight through it this
A:  Oh yeah.  It gives me a lot bigger perspective especially my kids and stu
```

## Chat QA with GPT-2

```
!pip install transformers
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pac
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /usr/local/lib
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/d
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-
```

```
    Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-p
    Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/
    Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-pacl
    Requirement already satisfied: tokenizers<0.19,>=0.14 in /usr/local/lib/pythor
    Requirement already satisfied: safetensors>=0.3.1 in /usr/local/lib/python3.1(
    Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-pa
    Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/(
    Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/py
    Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pytl
    Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-
    Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1(
    Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1(
```

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel
```

```
import torch
device = "cuda" if torch.cuda.is_available() else "cpu"

tokenizer = GPT2Tokenizer.from_pretrained("gpt2")
tokenizer.add_special_tokens({"pad_token": "<pad>",
                                "bos_token": "<startofstring>",
                                "eos_token": "<endofstring>"})
tokenizer.add_tokens(["<ans>:"])

model = GPT2LMHeadModel.from_pretrained("gpt2")
model.resize_token_embeddings(len(tokenizer))

model = model.to(device)
```

```
vocab.json: 100%                                  1.04M/1.04M [00:00<00:00, 7.08MB/s]

merges.txt: 100%                                  456k/456k [00:00<00:00, 4.42MB/s]

tokenizer.json:                                     1.36M/1.36M [00:00<00:00,
100%                                                15.1MB/s]

config.json: 100%                                 665/665 [00:00<00:00, 6.77kB/s]

model.safetensors:                                    548M/548M [00:06<00:00,
100%                                                  98.7MB/s]
```

```
from torch.utils.data import Dataset

class QAData(Dataset):
    def __init__(self, tokenizer, q, a):
        self.X = []
        for i in range(len(q)):
          self.X.append("<startofstring> "+ str(q[i]) +" <ans>: "+str(a[i])+" <en

        print(self.X[0])

        self.X_encoded = tokenizer(self.X, max_length=128, truncation=True, paddi
        self.X_encoded = tokenizer(self.X, max_length=128, truncation=True, paddi
        self.input_ids = self.X_encoded['input_ids']
        self.attention_mask = self.X_encoded['attention_mask']

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return (self.input_ids[idx], self.attention_mask[idx])

ds = QAData(tokenizer, all_questions, all_answers)
```

```
<startofstring>  You're back here in this place where you've won before.  Ind
```

```
import numpy as np
np.save('gpt2_ds',ds)
data = np.load('gpt2_ds')
```

```
/usr/local/lib/python3.10/dist-packages/numpy/lib/npyio.py:521: FutureWarning
  arr = np.asanyarray(arr)
/usr/local/lib/python3.10/dist-packages/numpy/lib/npyio.py:521: VisibleDeprec
  arr = np.asanyarray(arr)
```

```
from torch.utils.data import DataLoader
QAData =  DataLoader(ds, batch_size=16)
```

```
from tqdm import tqdm
import torch

def train(QAData, model, optim):

    epochs = 12

    for i in tqdm(range(epochs)):
        for X, a in QAData:
            X = X.to(device)
            a = a.to(device)
            optim.zero_grad()
            loss = model(X, attention_mask=a, labels=X).loss
            loss.backward()
            optim.step()
        torch.save(model.state_dict(), "model_state.pt")
        print(infer("how did you guys play today?"))

def infer(inp):
    inp = "<startofstring> "+inp+" <ans>: "
    inp = tokenizer(inp, return_tensors="pt")
    X = inp["input_ids"].to(device)
    a = inp["attention_mask"].to(device)
    output = model.generate(X, attention_mask=a )
    output = tokenizer.decode(output[0])
    return output

from torch.optim import Adam
model.train()

optim = Adam(model.parameters(), lr=1e-3)

print("training .... ")
train(QAData, model, optim)

print("infer from model : ")
while True:
  inp = input()
  print(infer(inp))
```

```
training ....
  0%|          | 0/12 [00:00<?, ?it/s]Setting `pad_token_id` to `eos_token_id
  8%|█         | 1/12 [00:13<02:26, 13.33s/it]
<startofstring> how did you guys play today?  <ans>:
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 17%|██        | 2/12 [00:25<02:08, 12.83s/it]
<startofstring> how did you guys play today?  <ans>:
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 25%|███       | 3/12 [00:38<01:54, 12.70s/it]
<startofstring> how did you guys play today?  <ans>:         I mean we
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 33%|███▍      | 4/12 [00:51<01:41, 12.71s/it]
<startofstring> how did you guys play today?  <ans>:        I don't think
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 42%|████▏     | 5/12 [01:03<01:29, 12.73s/it]
<startofstring> how did you guys play today?  <ans>:        I played five tim
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 50%|█████     | 6/12 [01:16<01:16, 12.73s/it]
<startofstring> how did you guys play today?  <ans>:       I mean they got a
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 58%|█████▊    | 7/12 [01:29<01:03, 12.76s/it]
<startofstring> how did you guys play today?  <ans>:      Yeah.  I played in
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 67%|██████▋   | 8/12 [01:41<00:50, 12.68s/it]
<startofstring> how did you guys play today?  <ans>:      I mean it was a big
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 75%|███████▌  | 9/12 [01:54<00:37, 12.66s/it]
<startofstring> how did you guys play today?  <ans>:     I played five times
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 83%|████████▎ | 10/12 [02:06<00:25, 12.58s/it]
<startofstring> how did you guys play today?  <ans>:     I played five years
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
 92%|█████████▏| 11/12 [02:19<00:12, 12.72s/it]
<startofstring> how did you guys play today?  <ans>:      I mean it's not a
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
100%|██████████| 12/12 [02:32<00:00, 12.70s/it]
<startofstring> how did you guys play today?  <ans>:      I played five years
```

```
infer from model :
Coach what do you think of the team's performance?
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
<startofstring> Coach what do you think of the team's performance?  <ans>:
Are you proud of the effort displayed today?
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
<startofstring> Are you proud of the effort displayed today?  <ans>:      I'm
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-12-685f3796ca7b> in <cell line: 10>()
      9 print("infer from model : ")
     10 while True:
---> 11   inp = input()
     12   print(infer(inp))

1 frames

/usr/local/lib/python3.10/dist-packages/ipykernel/kernelbase.py in
```

## BERT For NSP

```
from transformers import BertTokenizer, BertForNextSentencePrediction
import torch

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForNextSentencePrediction.from_pretrained('bert-base-uncased')

text = (all_questions[0])
text2 = (all_answers[0])

inputs = tokenizer(text, text2, return_tensors='pt')
inputs.keys()
```

```
tokenizer_config.json: 100%                 28.0/28.0 [00:00<00:00, 658B/s]
vocab.txt: 100%                  232k/232k [00:00<00:00, 1.39MB/s]
tokenizer.json: 100%                  466k/466k [00:00<00:00, 5.50MB/s]
config.json: 100%                  570/570 [00:00<00:00, 29.8kB/s]
model.safetensors:                  440M/440M [00:01<00:00,
```

```
inputs
```

```
{'input_ids': tensor([[  101,  2074,  2115,  4301,  2006,  1996,  2457,
2041,  2045,  1996,
          6891,  1998,  2115,  4301,  2006,  2652,  2122,  2399,  1996,
2345,
          2261,  2399,  1999,  5869,  7136,  1012,   102,  2026,  4301,
2006,
          1996,  2457,  2003,  2009,  3504,  2066,  1037,  2754,  1012,
2008,
          3504,  2079,  5051,  1012,  2023,  2878,  3325,  1045,  2228,
2009,
          1005,  1055,  1037,  2204,  9085,  9221,  2005,  2149,  2061,
2057,
          1005,  2128,  7568,  2005,  2023,  4990,  1012,  2057,  1005,
2128,
          7568,  2000,  2175,  2041,  2045,  1998,  5566,  1998, 11504,
2131,
          1996,  2663,  1012,   102]]), 'token_type_ids': tensor([[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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]), 'attention_mask': tensor([[1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
         1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]])}
```

```
labels = torch.LongTensor([0])
labels
```

```
tensor([0])
```

```
outputs = model(**inputs, labels=labels)
outputs.keys()
```

```
odict_keys(['loss', 'logits'])
```

```
outputs.loss
```

```
tensor(2.9802e-06, grad_fn=<NllLossBackward0>)
```

```
outputs.loss.item()
```

```
2.9802276912960224e-06
```

```
together = []
for i in (range(len(all_questions))):
  together.append(all_questions[i])
  together.append(all_answers[i])
```

```
import random

sentence_a = []
sentence_b = []
sentence_b = []
label = []

for i in (range(len(all_questions))):
  # 50/50 whether is IsNextSentence or NotNextSentence
  if random.random() >= 0.2:
      # this is IsNextSentence
      sentence_a.append(all_questions[i])
      sentence_b.append(all_answers[i])
      label.append(0)
  else:
      index = random.randint(0, len(all_questions)-1)
      # this is NotNextSentence
      sentence_a.append(all_questions[i])
      sentence_b.append(all_answers[index])
      label.append(1)

for i in range(3):
    print(label[i])
    print(sentence_a[i] + '\n---')
    print(sentence_b[i] + '\n')
```

```
0
Just your thoughts on the court out there the venue and your thoughts on play
---
   My thoughts on the court is it looks like a stage.  That looks dope.  This

0
BI you just mentioned how you guys aren't on national TV a lot.  CJ reference
---
   Goofy.  We come to work we all have fun.  Off the court on the court we al

0
You mentioned that you guys like to have fun.  Have you thought about how you
---
   No we ain't thought about it but I know it's going to be a good time.
```

```
inputs = tokenizer(sentence_a, sentence_b, return_tensors='pt', max_length=512, t
```

```
inputs.keys()
```

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

```
inputs.token_type_ids[0]
```

```
tensor([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, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1,
        1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
```

```
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 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,
                 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, 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, 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])
```

```
inputs['labels'] = torch.LongTensor([label]).T
```

```
inputs.labels[:10]
```

```
    tensor([[0],
            [0],
            [0],
            [0],
            [0],
            [0],
            [1],
            [1],
            [0],
            [0]])
```

```
class QADataset(torch.utils.data.Dataset):
    def __init__(self, encodings):
        self.encodings = encodings
    def __getitem__(self, idx):
        return {key: torch.tensor(val[idx]) for key, val in self.encodings.items(
    def __len__(self):
        return len(self.encodings.input_ids)

dataset = QADataset(inputs)


loader = torch.utils.data.DataLoader(dataset, batch_size=8, shuffle=True)


device = torch.device('cuda') if torch.cuda.is_available() else torch.device('cpu
# and move our model over to the selected device
model.to(device)
```

```
BertForNextSentencePrediction(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(30522, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (cls): BertOnlyNSPHead(
    (seq_relationship): Linear(in_features=768, out_features=2, bias=True)
  )
)
```

```
from transformers import AdamW

model.train()

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

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: Fut
  warnings.warn(
```

```
import matplotlib.pyplot as plt

from tqdm import tqdm  # for our progress bar

epochs = 5
y = []
for epoch in range(epochs):
    # setup loop with TQDM and dataloader
    loop = tqdm(loader, leave=True)
    for batch in loop:
        # initialize calculated gradients (from prev step)
        optim.zero_grad()
        # pull all tensor batches required for training
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        token_type_ids = batch['token_type_ids'].to(device)
        labels = batch['labels'].to(device)
        # process
        outputs = model(input_ids, attention_mask=attention_mask,
                        token_type_ids=token_type_ids,
                        labels=labels)
        # extract loss
        loss = outputs.loss
        # calculate loss for every parameter that needs grad update
        loss.backward()
        # update parameters
        optim.step()
        # print relevant info to progress bar
        loop.set_description(f'Epoch {epoch}')
        loop.set_postfix(loss=loss.item())
        y.append(float(loss.item()))
        y.append(float(loss.item()))

plt.xlabel('Epochs')
plt.ylabel('Loss Per Epoch')
plt.title('Loss')

x = list(range(1, epochs + 1))

plt.plot(x, y)

plt.show()
```

```
   0%|          | 0/130 [00:00<?, ?it/s]<ipython-input-10-834994efa95d>:5: Use
   return {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
  Epoch 0: 100%|██████████| 130/130 [00:32<00:00,  4.00it/s, loss=0.205]
  Epoch 1: 100%|██████████| 130/130 [00:30<00:00,  4.22it/s, loss=0.00706]
  Epoch 2: 100%|██████████| 130/130 [00:31<00:00,  4.08it/s, loss=0.0711]
  Epoch 3: 100%|██████████| 130/130 [00:31<00:00,  4.18it/s, loss=0.0181]
  Epoch 4: 100%|██████████| 130/130 [00:30<00:00,  4.20it/s, loss=0.00349]
```

```
# Regular BERT model for comparison
bert_model = BertForNextSentencePrediction.from_pretrained('bert-base-uncased')

bert_model = bert_model.to(device)

# Text input
input_text = "Do you learn something from it?"
next_sentence = "Yeah I think you learn"

# Tokenize the input
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
inputs = tokenizer(input_text, next_sentence, return_tensors='pt')

inputs = inputs.to(device)

# Get predictions from your fine-tuned model
with torch.no_grad():
    outputs = model(**inputs)
    prediction = "IsNext" if outputs.logits[0][0] > outputs.logits[0][1] else "No

    # Decode the input IDs to get the actual sentences
    decoded_input = tokenizer.decode(inputs['input_ids'][0], skip_special_tokens=
    decoded_next_sentence = tokenizer.decode(inputs['input_ids'][0], skip_special

# Get predictions from the regular BERT model
with torch.no_grad():
    outputs_bert = bert_model(**inputs)
    prediction_bert = "IsNext" if outputs_bert.logits[0][0] > outputs_bert.logits

    # Decode the input IDs to get the actual sentences
    decoded_input_bert = tokenizer.decode(inputs['input_ids'][0], skip_special_to
    decoded_next_sentence_bert = tokenizer.decode(inputs['input_ids'][0], skip_sp
    decoded_next_sentence_bert = tokenizer.decode(inputs['input_ids'][0], skip_sp

# Print the actual sentences and predictions
print(f"Fine-tuned Model Prediction: {prediction}")
print(f"Fine-tuned Model Input: {decoded_input}")
print(f"Fine-tuned Model Next Sentence: {decoded_next_sentence}")

print(f"\nRegular BERT Model Prediction: {prediction_bert}")
print(f"Regular BERT Model Input: {decoded_input_bert}")
print(f"Regular BERT Model Next Sentence: {decoded_next_sentence_bert}")
```

```
Fine-tuned Model Prediction: IsNext
Fine-tuned Model Input: do you learn something from it? yeah i think you learn
Fine-tuned Model Next Sentence: do you learn something from it? yeah i think y

Regular BERT Model Prediction: IsNext
Regular BERT Model Input: do you learn something from it? yeah i think you lea
Regular BERT Model Next Sentence: do you learn something from it? yeah i think
```

## FAILED ATTEMPT: T5 Question and Answering

```
!nvidia-smi
```

```
Wed Dec 20 16:45:04 2023
+------------------------------------------------------------------------------
| NVIDIA-SMI 535.104.05             Driver Version: 535.104.05   CUDA Version
|-----------------------------------------+----------------------+------------
| GPU  Name                 Persistence-M | Bus-Id        Disp.A | Volatile Un
| Fan  Temp   Perf          Pwr:Usage/Cap |         Memory-Usage | GPU-Util  (
|                                         |                      |
|=========================================+======================+============
|   0  Tesla T4                       Off | 00000000:00:04.0 Off |
| N/A   69C    P8              11W /  70W |      0MiB / 15360MiB |      0%
|                                         |                      |
+-----------------------------------------+----------------------+------------

+------------------------------------------------------------------------------
| Processes:
|  GPU   GI   CI        PID   Type   Process name                            (
|        ID   ID                                                             U
|==============================================================================
|  No running processes found
+------------------------------------------------------------------------------
```

```
!pip install transformers==4.1.1
!pip install pytorch-lightning==1.1.1
!pip install tokenizers==0.9.4
```

```
!pip install sentencepiece==0.1.94
```

```
import torchtext
!pip install torchtext.data
```

```
    ERROR: Could not find a version that satisfies the requirement torchtext.data
    ERROR: No matching distribution found for torchtext.data
```

```
#import pytorch_lightning as pl
import re
import pandas as pd
import numpy as np
from transformers import (AdamW, T5ForConditionalGeneration, T5Tokenizer, get_lin
```

```
tokenizer = T5Tokenizer.from_pretrained("t5-base")
```

```
    spiece.model:   0%|          | 0.00/792k [00:00<?, ?B/s]
    tokenizer.json:                                   1.39M/1.39M [00:00<00:00,
    100%                                              5.40MB/s]
    config.json: 100%                                 1.21k/1.21k [00:00<00:00, 70.5kB/s]
    /usr/local/lib/python3.10/dist-packages/transformers/models/t5/tokenization_t!
    For now, this behavior is kept to avoid breaking backwards compatibility when
    - Be aware that you SHOULD NOT rely on t5-base automatically truncating your :
    - If you want to encode/pad to sequences longer than 512 you can either instal
    - To avoid this warning, please instantiate this tokenizer with `model_max_lei
      warnings.warn(
    You are using the default legacy behaviour of the <class 'transformers.models
    Special tokens have been added in the vocabulary, make sure the associated wo
```

```
sample = tokenizer("This is just trying this out", "another try at this")
```

```
sample_q = "trying this out again, will it work?"
```

```
sample.keys()
```

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

```
print(sample["input_ids"])
```

```
    [100, 19, 131, 1119, 48, 91, 1, 430, 653, 44, 48, 1]
```

```
print(sample["attention_mask"])
```

```
    [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

```
pred = [tokenizer.decode(input, skip_special_tokens=True, clean_up_tokenization_s
```

```
" ".join(pred)
```

```
.join(pred)
```

```
'This is just trying this out </s> another try at this </s>'
```

```
dict = {'questions': all_questions, 'answers': all_answers}

df = pd.DataFrame(dict)

print(df)
```

```
                                              questions  \
0     Just your thoughts on the court out there the ...
1     BI you just mentioned how you guys aren't on n...
2     You mentioned that you guys like to have fun. ...
3     CJ had mentioned the prize money a couple time...
4     What's the best and worst thing you can say ab...
...                                                 ...
1035  You had many ups and downs this season.  What'...
1036  George can you talk about the long touchdown r...
1037  George and Andrew I'm curious this last month ...
1038  George probably played his last game as a Bron...
1039  How do you want this team to be remembered?  H...

                                                answers
0        My thoughts on the court is it looks like a...
1        Goofy.  We come to work we all have fun.  O...
2        No we ain't thought about it but I know it'...
3        For me I think my motivating factor is just...
4        I don't know you caught me off guard with t...
...                                                 ...
1035     Definitely my favorite part of the journey ...
1036     Yes in the first half we kicked a couple of...
1037     You know how strong the team is the perseve...
1038     George's legacy has been imprinted for a lo...
1039     I'm excited.  I believe God is up to someth...

[1040 rows x 2 columns]
```

```
import torch
from torch.utils.data import Dataset, DataLoader
from sklearn.model_selection import train_test_split

class t5ds(Dataset):

  def __init__(self, data, tokenizer: T5Tokenizer, q_len: int = 396,a_len: int =

    self.tokenizer = tokenizer
    self.data = data
    self.q_len = q_len
    self.a_len = a_len

  def __len__(self):
    return len(self.data)
  def __getitem__(self, idx):
    row = self.data.iloc[idx]

    q_enc = self.tokenizer(row["questions"], max_length = self.q_len, padding="ma
    a_enc = self.tokenizer(row["answers"], max_length = self.a_len, padding="max_

    labels = a_enc["input_ids"]
    labels[labels == 0] = -100

    return {"question": row["questions"], "answer_text": row["answers"], "input_i

sample_dataset = t5ds(df, tokenizer)


for data in sample_dataset:
  print(data)
  break
```

```
     {'question': 'Just your thoughts on the court out there the venue and your th
            3376,   30, 1556,  175, 1031,    8,  804,  360, 1031,   16, 7263, 761
               5,    1,    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,    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,
               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,    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,    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,
            1, 1, 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, 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, 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, (
            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, 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]), 'labels': [tensor(499), tensor(
```

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=.1)
```

```
train.shape, test.shape
```

```
    ((936, 2), (104, 2))
```

```
!pip install pytorch-lightning==1.5.10
!pip install omegaconf -U
!pip install hydra-core --upgrade
import pytorch_lightning as pl
print(pl.__version__)
```

```
class t5DataModule(pl.LightningDataModule):
  def __init__(self, train_df, test_df, tokenizer, batch_size = 8, q_max: int = 3
    super().__init__()
    self.batch_size = batch_size
    self.test_df = test_df
    self.train_df = train_df
    self.tokenizer = tokenizer
    self.q_max = q_max
    self.a_max = a_max

  def setup(self):
    self.train_ds = t5ds(self.train_df, self.tokenizer, self.q_max, self.a_max)
    self.test_ds = t5ds(self.test_df, self.tokenizer, self.q_max, self.a_max)

  def train_dataloader(self):
     return DataLoader(self.train_ds, batch_size = self.batch_size, shuffle=True,

  def val_dataloader(self):
     return DataLoader(self.train_ds, batch_size = 1, num_workers=4)

  def test_dataloader(self):
     return DataLoader(self.test_ds, batch_size = 1, num_workers=4)
```

```
BATCH_SIZE = 2
N_EPOCHS = 4

data_module = t5DataModule(train, test, tokenizer, batch_size = BATCH_SIZE)
data_module.setup()

model = T5ForConditionalGeneration.from_pretrained('t5-base', return_dict=True)
```

```
model.config
```

```
T5Config {
  "_name_or_path": "t5-base",
  "architectures": [
    "T5ForConditionalGeneration"
  ],
  "classifier_dropout": 0.0,
  "d_ff": 3072,
  "d_kv": 64,
  "d_model": 768,
  "decoder_start_token_id": 0,
  "dense_act_fn": "relu",
  "dropout_rate": 0.1,
  "eos_token_id": 1,
  "feed_forward_proj": "relu",
  "initializer_factor": 1.0,
  "is_encoder_decoder": true,
  "is_gated_act": false,
  "layer_norm_epsilon": 1e-06,
  "model_type": "t5",
  "n_positions": 512,
  "num_decoder_layers": 12,
  "num_heads": 12,
  "num_layers": 12,
  "output_past": true,
  "pad_token_id": 0,
  "relative_attention_max_distance": 128,
  "relative_attention_num_buckets": 32,
  "task_specific_params": {
    "summarization": {
      "early_stopping": true,
      "length_penalty": 2.0,
      "max_length": 200,
      "min_length": 30,
      "no_repeat_ngram_size": 3,
      "num_beams": 4,
      "prefix": "summarize: "
    },
    "translation_en_to_de": {
      "early_stopping": true,
      "max_length": 300,
      "num_beams": 4,
      "prefix": "translate English to German: "
    },
    "translation_en_to_fr": {
      "early_stopping": true,
      "max_length": 300,
      "num_beams": 4,
      "prefix": "translate English to French: "
    },
    "translation_en_to_ro": {
      "early_stopping": true,
      "max_length": 300,
      "num_beams": 4,
      "prefix": "translate English to Romanian: "
    }
  },
  "transformers_version": "4.35.2",
  "use_cache": true,
```

```
class newT5(pl.LightningModule):
  def __init__(self):
    super().__init__()
    self.model = T5ForConditionalGeneration.from_pretrained('t5-base', return_dic

  def forward(self, input_ids, attention_mask, labels=None):
    output = model(input_ids=input_ids, attention_mask=attention_mask, labels=lab
    if labels is not None:
      return output.loss, output.logits
    else:
      return output.logits
    return output.loss, output.logits

  def training_step(self, batch, batch_idx):
    input_ids = batch["input_ids"]
    attention_mask = batch["attention_mask"]
    labels = batch["labels"]
    loss, outputs = self(input_ids, attention_mask, labels)
    self.log("train_loss", loss, prog_bar=True, logger=True)
    return loss

  def validation_step(self, batch, batch_idx):
    input_ids = batch["input_ids"]
    attention_mask = batch["attention_mask"]
    labels = batch["labels"]
    loss, outputs = self(input_ids, attention_mask, labels)
    self.log("val_loss", loss, prog_bar=True, logger=True)
    return loss

  def test_step(self, batch, batch_idx):
    input_ids = batch["input_ids"]
    attention_mask = batch["attention_mask"]
    labels = batch["labels"]
    loss, outputs = self(input_ids, attention_mask, labels)
    self.log("test_loss", loss, prog_bar=True, logger=True)
    self.log("test_loss", loss, prog_bar=True, logger=True)
    return loss

  def configure_optimizers(self):
    return AdamW(self.parameters(), lr=.0001)
```

```
model = newT5()
```

```
from keras.callbacks import ModelCheckpoint
checkpoint_callback = ModelCheckpoint(filepath="checkpoints", filename="best_chec
```

```
trainer = pl.Trainer(checkpoint_callback=checkpoint_callback, max_epochs = 3, gpu
```

```
    /usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/connectors/
      rank_zero_deprecation(
    /usr/local/lib/python3.10/dist-packages/pytorch_lightning/trainer/connectors/
      rank_zero_deprecation(
    INFO:pytorch_lightning.utilities.distributed:GPU available: True, used: True
    INFO:pytorch_lightning.utilities.distributed:TPU available: False, using: 0 T
    INFO:pytorch_lightning.utilities.distributed:IPU available: False, using: 0 I
```

```
%load_ext tensorboard
```

```
    The tensorboard extension is already loaded. To reload it, use:
      %reload_ext tensorboard
```

```
%tensorboard --logdir ./lightning_logs
```

TensorBoard INACTIVE

## No dashboards are active for the current data set.

Probable causes:

- You haven't written any data to your event files.
- TensorBoard can't find your event files.

If you're new to using TensorBoard, and want to find out how to add data and set up your event files, check out the README and perhaps the TensorBoard tutorial.

If you think TensorBoard is configured properly, please see the section of the README devoted to missing data problems and consider filing an issue the README devoted to missing data problems and consider filing an issue on GitHub.

*Last reload: Dec 23, 2023, 10:33:25 PM*

*Log directory: ./lightning_logs*

```
trainer.fit(model, data_module)
```

```
/usr/local/lib/python3.10/dist-packages/pytorch_lightning/core/datamodule.py:4
  rank_zero_deprecation(
INFO:pytorch_lightning.accelerators.gpu:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES:
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:411: Futu
  warnings.warn(
INFO:pytorch_lightning.callbacks.model_summary:
  | Name  | Type                       | Params
-----------------------------------------------------
0 | model | T5ForConditionalGeneration | 222 M
-----------------------------------------------------
222 M     Trainable params
0         Non-trainable params
222 M     Total params
891.614   Total estimated model params size (MB)
```

Validation sanity check: 0% 0/2 [00:00<?, ?it/s]

```
/usr/local/lib/python3.10/dist-packages/torch/utils/data/dataloader.py:557: Us
  warnings.warn(_create_warning_msg(
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
<ipython-input-61-7b6b8391c42e> in <cell line: 1>()
----> 1 trainer.fit(model, data_module)
```

20 frames

```
... last 3 frames repeated, from the frame below ...
```

```
/usr/local/lib/python3.10/dist-packages/torch/nn/modules/module.py in
_wrapped_call_impl(self, *args, **kwargs)
   1516             return self._compiled_call_impl(*args, **kwargs)  # type:
ignore[misc]
   1517         else:
-> 1518             return self._call_impl(*args, **kwargs)
   1519 
   1520     def _call_impl(self, *args, **kwargs):

RecursionError: maximum recursion depth exceeded while calling a Python
object
```

SEARCH STACK OVERFLOW

## RNN for Q and A

```
import nltk
nltk.download('punkt')
from nltk.tokenize import word_tokenize

token_q = []
token_a = []

for sent in all_questions:
  token_q.append(word_tokenize(sent))

for sent in all_answers:
  token_a.append(word_tokenize(sent))
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

```
print(token_q[0])
print(token_a[0])
```

```
['Just', 'your', 'thoughts', 'on', 'the', 'court', 'out', 'there', 'the', 'ver
['My', 'thoughts', 'on', 'the', 'court', 'is', 'it', 'looks', 'like', 'a', 's
```

```
def maxi(sents, others):
    for x in range(100):
        nmax = 0
        to_remove = None
        for sent in sents:
            if len(sent) > nmax:
                nmax = len(sent)
                to_remove = sents.index(sent)
        del sents[to_remove]
        del others[to_remove]
    return others, sents
```

```
def bracket_questions(sent):
    sent = ['<q>'] + sent + ['</q>']
    return sent

def bracket_answer(sent):
    sent = ['<a>'] + sent + ['</a>']
    return sent

print(len(token_q))
print(len(token_a))
token_q, token_a = maxi(token_a, token_q)
print(len(token_q))
print(len(token_a))

for x in range(len(token_q)):
    token_q[x] = bracket_questions(token_q[x])
    token_a[x] = bracket_answer(token_a[x])
```

```
1040
1040
940
940
```

```
from google.colab import drive
drive.mount('/content/drive')

data_dir = 'drive/MyDrive/'
```

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

```
print(token_q[0])
print(token_a[0])
```

```
['<q>', 'Just', 'your', 'thoughts', 'on', 'the', 'court', 'out', 'there', 'th
['<a>', 'My', 'thoughts', 'on', 'the', 'court', 'is', 'it', 'looks', 'like',
```

```
vocab = {}
i = 0
for sent in token_q:
    for token in sent:
        if token not in vocab:
          vocab[token] = i
          i += 1

for sent in token_a:
    for token in sent:
        if token not in vocab:
          vocab[token] = i
          i += 1
```

```
idx_to_token = {idx: token for token, idx in vocab.items()}

print(vocab)


import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np

def load_glove_embeddings(glove_file):
    weights_matrix = torch.zeros(len(vocab), 100)
    embeddings = {}
    with open(glove_file, 'r', encoding='utf-8') as file:
        for line in file:
            values = line.split()
            word = values[0]
            vector = np.array([float(val) for val in values[1:]])
            embeddings[word] = vector
            if word in vocab:
              idx = vocab[word]
              weights_matrix[idx] = torch.tensor(np.asarray(values[1:], "float32"
    for word, idx in vocab.items():
      if not torch.any(weights_matrix[idx]):
        weights_matrix[idx] = torch.normal(0.0, 0.6, size=(100, ))

    return weights_matrix, embeddings

weights_matrix, glove_embeddings = load_glove_embeddings(data_dir + './glove.6B.1

all_tokens = []
for i in range(len(token_q)):
  for token in token_q[i]:
    if token == 'J':
      print(all_questions[i])
    all_tokens.append(token)
  for token in token_a[i]:
    if token == 'J':
      print(all_answers[i])
    all_tokens.append(token)

encode = np.vectorize(lambda w: vocab[w])
encoded = encode(all_tokens)

sequences = []
goals = []

for i in range(len(encoded) - 41):
    sequences.append((encoded[i:i+40]))
    goals.append((encoded[i+1:i+41]))
    goals.append((encoded[i+1:i+41]))
print(sequences[0])
print(goals[0])
```

```
[   0    1    2    3    4    5    6    7    8    5    9   10    2    3
    4   11   12   13    5   14   15   13   16   17   18   19   20 3186
 3187    3    4    5    6   88   91 1557   62   31  213   19]
[   1    2    3    4    5    6    7    8    5    9   10    2    3    4
   11   12   13    5   14   15   13   16   17   18   19   20 3186 3187
    3    4    5    6   88   91 1557   62   31  213   19 1586]
```

```
class NEW_DS(Dataset):

  def __init__(self, X, Y):
        self.X = X  # Convert X to a PyTorch LongTensor
        self.Y = Y

  def __len__(self):
      return len(self.X)

  def __getitem__(self, idx):

      return torch.tensor(self.X[idx]), torch.tensor(self.Y[idx])


batch_size = 32
dataset = NEW_DS(sequences, goals)

data_loader = DataLoader(dataset, batch_size = 32, shuffle=False)

class MyLSTM(nn.Module):
    def __init__(self, vocab_size, embedding_size, hidden_size, embedding_weights
        super(MyLSTM, self).__init__()
        self.embedding = nn.Embedding(vocab_size, embedding_size)
        self.embedding.weight = nn.Parameter(embedding_weights)
        self.embedding.weight.requires_grad = True
        self.lstm = nn.LSTM(embedding_size, hidden_size, batch_first=True)
        self.linear = nn.Linear(hidden_size, vocab_size)

    def forward(self, x):
        embedded = self.embedding(x)
        lstm_out, _ = self.lstm(embedded)
        output = self.linear(lstm_out)
        return output

import matplotlib.pyplot as plt

vocab_size = len(vocab)
embedding_size = 100
hidden_size = 256
```

```
model = MyLSTM(vocab_size, embedding_size, hidden_size, weights_matrix)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 4
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
y = []
for epoch in range(num_epochs):
    model.train()
    total_loss = 0

    for batch_idx, (inputs, targets) in enumerate(data_loader):
        inputs, targets = inputs.to(device), targets.to(device)

        optimizer.zero_grad()
        outputs = model(inputs)

        loss = criterion(outputs.permute(0, 2, 1), targets)  # Permute outputs to
        total_loss += loss.item()

        loss.backward()
        optimizer.step()

        if batch_idx % 100 == 0:
            print(f'Epoch [{epoch+1}/{num_epochs}], Batch [{batch_idx+1}/{len(dat

    average_loss = total_loss / len(data_loader)
    print(f'Epoch [{epoch+1}/{num_epochs}], Average Loss: {average_loss:.4f}')
    y.append(average_loss)

plt.xlabel('Epochs')
plt.ylabel('Avg Loss Per Epoch')
plt.title('Loss')

x = list(range(1, num_epochs + 1))

plt.plot(x, y)

plt.show()
```

```
Epoch [1/4], Batch [1/2553], Loss: 8.4941
Epoch [1/4], Batch [101/2553], Loss: 6.0208
Epoch [1/4], Batch [201/2553], Loss: 6.0986
Epoch [1/4], Batch [301/2553], Loss: 6.0233
Epoch [1/4], Batch [401/2553], Loss: 5.3439
Epoch [1/4], Batch [501/2553], Loss: 4.9774
Epoch [1/4], Batch [601/2553], Loss: 5.0792
Epoch [1/4], Batch [701/2553], Loss: 5.1037
Epoch [1/4], Batch [801/2553], Loss: 5.2506
Epoch [1/4], Batch [901/2553], Loss: 5.0924
Epoch [1/4], Batch [1001/2553], Loss: 4.9033
Epoch [1/4], Batch [1101/2553], Loss: 5.4155
Epoch [1/4], Batch [1201/2553], Loss: 4.6104
Epoch [1/4], Batch [1301/2553], Loss: 4.4934
Epoch [1/4], Batch [1401/2553], Loss: 4.8183
Epoch [1/4], Batch [1501/2553], Loss: 4.4867
Epoch [1/4], Batch [1601/2553], Loss: 4.4682
Epoch [1/4], Batch [1701/2553], Loss: 4.3999
Epoch [1/4], Batch [1801/2553], Loss: 4.4497
Epoch [1/4], Batch [1901/2553], Loss: 5.8747
Epoch [1/4], Batch [2001/2553], Loss: 5.1210
Epoch [1/4], Batch [2101/2553], Loss: 4.6843
Epoch [1/4], Batch [2201/2553], Loss: 6.1195
Epoch [1/4], Batch [2301/2553], Loss: 4.0410
Epoch [1/4], Batch [2401/2553], Loss: 4.5573
Epoch [1/4], Batch [2501/2553], Loss: 4.6640
Epoch [1/4], Average Loss: 5.1581
Epoch [2/4], Batch [1/2553], Loss: 4.5541
Epoch [2/4], Batch [101/2553], Loss: 4.3439
Epoch [2/4], Batch [201/2553], Loss: 4.3460
Epoch [2/4], Batch [301/2553], Loss: 4.5638
Epoch [2/4], Batch [401/2553], Loss: 3.8549
Epoch [2/4], Batch [501/2553], Loss: 4.0581
Epoch [2/4], Batch [601/2553], Loss: 4.0330
Epoch [2/4], Batch [701/2553], Loss: 4.0717
Epoch [2/4], Batch [801/2553], Loss: 4.5112
Epoch [2/4], Batch [901/2553], Loss: 4.3105
Epoch [2/4], Batch [1001/2553], Loss: 4.2491
Epoch [2/4], Batch [1101/2553], Loss: 4.5146
Epoch [2/4], Batch [1201/2553], Loss: 4.0246
Epoch [2/4], Batch [1301/2553], Loss: 3.6104
Epoch [2/4], Batch [1401/2553], Loss: 3.9061
Epoch [2/4], Batch [1501/2553], Loss: 3.7914
Epoch [2/4], Batch [1601/2553], Loss: 3.9098
Epoch [2/4], Batch [1701/2553], Loss: 3.6242
Epoch [2/4], Batch [1801/2553], Loss: 3.9324
Epoch [2/4], Batch [1901/2553], Loss: 4.6476
Epoch [2/4], Batch [2001/2553], Loss: 4.6767
Epoch [2/4], Batch [2101/2553], Loss: 4.2307
Epoch [2/4], Batch [2201/2553], Loss: 5.2817
Epoch [2/4], Batch [2301/2553], Loss: 3.6146
Epoch [2/4], Batch [2401/2553], Loss: 4.0233
Epoch [2/4], Batch [2501/2553], Loss: 4.0975
Epoch [2/4], Average Loss: 4.2554
Epoch [3/4], Batch [1/2553], Loss: 3.9095
Epoch [3/4], Batch [101/2553], Loss: 4.0255
Epoch [3/4], Batch [201/2553], Loss: 3.8844
Epoch [3/4], Batch [301/2553], Loss: 4.0980
```

```
model.eval()

# Define a starting sequence for prediction
start_sequence = ['<q>','how', 'was', 'the', 'game', '</q>', '<a>']
```

```
    # Encode the starting sequence to integers
    encoded_start_sequence = [vocab[char] for char in start_sequence]

    # Convert the encoded start sequence to a PyTorch tensor
    inputs = torch.tensor(encoded_start_sequence).unsqueeze(0)  # Add batch dimension

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    inputs = inputs.to(device)

    # Predict the next character
    with torch.no_grad():
        hidden = None
        for i in range(20):  # Predict the next 50 characters
            #print(inputs.shape)
            outputs = model(inputs)
            # Get the output logits for the last character in the sequence
            last_output = outputs[:, -1, :]
            # Get the index of the predicted character
            predicted_index = torch.argmax(last_output, dim=1).item()
            # Map the index back to the character
            predicted = idx_to_token[predicted_index]
            # Print the predicted character
            if predicted == '</a>':
              break
            print(" " + predicted + " ", end='')

            # Update the input sequence for the next prediction
            inputs = torch.cat((inputs, torch.tensor([[predicted_index]]).to(device))
```

```
 Yeah  .
```