In [ ]:
```python
from transformers import AutoConfig, AutoModelForCausalLM, AutoTokenizer
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F
import torch
import torch
import classifier
from tokenizer import BertTokenizer
from bert import BertModel
from tqdm import tqdm

from transformers import AutoTokenizer, AutoModelForCausalLM
import numpy as np
```

4.1

In [ ]:
```python
#Removed args from BertDataset in classifier.py
class BertDataset(Dataset):
    def __init__(self, dataset):
        self.dataset = dataset
        self.tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

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

    def __getitem__(self, idx):
        ele = self.dataset[idx]
        return ele

    def pad_data(self, data):
        sents = [x[0] for x in data]
        labels = [x[1] for x in data]
        encoding = self.tokenizer(sents, return_tensors='pt', padding=True, truncat
        token_ids = torch.LongTensor(encoding['input_ids'])
        attention_mask = torch.LongTensor(encoding['attention_mask'])
        token_type_ids = torch.LongTensor(encoding['token_type_ids'])
        labels = torch.LongTensor(labels)

        return token_ids, token_type_ids, attention_mask, labels, sents

    def collate_fn(self, all_data):
        print("collated")
        token_ids, token_type_ids, attention_mask, labels, sents = self.pad_data(al
        batched_data = {
                'token_ids': token_ids,
                'token_type_ids': token_type_ids,
                'attention_mask': attention_mask,
                'labels': labels,
                'sents': sents,
            }

        return batched_data
```

In [ ]:
```
#Code simplified from test() in classifier.py

#Using CPU
device = torch.device('cpu')

#Load Model from 3.2
saved = torch.load("flexible-10-1e-05.pt", map_location=device)
config = saved['model_config']
model = classifier.BertSentClassifier(config)
model.load_state_dict(saved['model'])
model = model.to(device)


dev_data = classifier.create_data("reviews.txt", 'test')
dev_dataset = BertDataset(dev_data)
dev_dataloader = DataLoader(dev_dataset, shuffle=False, batch_size=5, collate_fn=de
dev_acc, dev_f1, dev_pred, dev_true, dev_sents = classifier.model_eval(dev_dataload

with open("./reviews_output.txt", "w+") as f:
    print(f"test acc :: {dev_acc :.3f}")
    for s, p in zip(dev_sents, dev_pred):
        f.write(f"{p} ||| {s}\n")
```

```
load 5 data from reviews.txt
eval:   0%|          | 0/1 [00:00<?, ?it/s]
collated
eval: 100%|██████████| 1/1 [01:50<00:00, 110.22s/it]
tensor([[0.0060, 0.9940],
        [0.0108, 0.9892],
        [0.1384, 0.8616],
        [0.9894, 0.0106],
        [0.4132, 0.5868]], grad_fn=<SoftmaxBackward0>)
test acc :: 0.800
```

Comments

The model performed ok, it was able to successfully determine the very positive/negative reviews, but struggled with the less negative review. It determined that the random sentence was a positive review. Overall it matches my expectations, the important thing is that it was able to classify the very positive/negative sentences correctly.

4.2

In [ ]:
```
#Setting the seeds
seed = 1004804651

torch.manual_seed(seed)
torch.cuda.manual_seed(seed)
torch.cuda.manual_seed_all(seed)
```

```
torch.backends.cudnn.benchmark = False
torch.backends.cudnn.deterministic = True
```

In [ ]:
```
#Creates an AutoModelForCausalLM from transformers
config = AutoConfig.from_pretrained("bert-base-uncased")
modelLM = AutoModelForCausalLM.from_config(config)
```

```
If you want to use `BertLMHeadModel` as a standalone, add `is_decoder=True.`
```

In [ ]:
```
#Ordered by very positive, less positive, less negative, very negative, and random
sentences = ["I really loved this movie! The acting was phenominal and the cinemato
             "This movie was alright, I just liked the ending.",
             "The acting could have been better.",
             "This movie was a waste of my time, I really disliked everything about
             "Toronto is a city in Ontario, Canada."]
```

In [ ]:
```
tokenizer = AutoTokenizer.from_pretrained("bert-base-uncased")
tokenzied_sentences = []
for sentence in sentences:
    #Appends "This movie review is" to each sentence
    new_sentence = sentence + " This polarity of the sentence is"

    #Tokenzies the sentences
    tokenized_sentences.append(tokenizer(new_sentence,return_tensors='pt',padding='
```

In [ ]:
```
#Creates lists of IDs and attention masks for each review
id_list = []
attention_mask_list = []
for sentence in tokenized_sentences:
    ids = torch.LongTensor(sentence['input_ids'])
    attention_mask = torch.LongTensor(sentence['attention_mask'])

    print(ids)
    id_list.append(ids)
    attention_mask_list.append(attention_mask)
```

```
tensor([[  101,  1045,  2428,  3866,  2023,  3185,   999,  1996,  3772,  2001,
          6887, 16515, 22311,  2140,  1998,  1996, 16434,  2001,  6429,  1012,
           102,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[  101,  2023,  3185,  2001, 10303,  1010,  1045,  2074,  4669,  1996,
          4566,  1012,   102,     0,     0,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[ 101, 1996, 3772, 2071, 2031, 2042, 2488, 1012,  102,    0,    0,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0]])
tensor([[  101,  2023,  3185,  2001,  1037,  5949,  1997,  2026,  2051,  1010,
          1045,  2428, 18966,  2673,  2055,  2009,  1012,   102,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[ 101, 4361, 2003, 1037, 2103, 1999, 4561, 1010, 2710, 1012,  102,    0,
            0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0]])
tensor([[  101,  1045,  2428,  3866,  2023,  3185,   999,  1996,  3772,  2001,
          6887, 16515, 22311,  2140,  1998,  1996, 16434,  2001,  6429,  1012,
          2023,  3185,  3319,  2003,   102,     0,     0,     0,     0,     0]])
tensor([[  101,  2023,  3185,  2001, 10303,  1010,  1045,  2074,  4669,  1996,
          4566,  1012,  2023,  3185,  3319,  2003,   102,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[ 101, 1996, 3772, 2071, 2031, 2042, 2488, 1012, 2023, 3185, 3319, 2003,
          102,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0]])
tensor([[  101,  2023,  3185,  2001,  1037,  5949,  1997,  2026,  2051,  1010,
          1045,  2428, 18966,  2673,  2055,  2009,  1012,  2023,  3185,  3319,
          2003,   102,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[ 101, 4361, 2003, 1037, 2103, 1999, 4561, 1010, 2710, 1012, 2023, 3185,
         3319, 2003,  102,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0]])
tensor([[  101,  1045,  2428,  3866,  2023,  3185,   999,  1996,  3772,  2001,
          6887, 16515, 22311,  2140,  1998,  1996, 16434,  2001,  6429,  1012,
          2023,  3185,  3319,  2003,   102,     0,     0,     0,     0,     0]])
tensor([[  101,  2023,  3185,  2001, 10303,  1010,  1045,  2074,  4669,  1996,
          4566,  1012,  2023,  3185,  3319,  2003,   102,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[ 101, 1996, 3772, 2071, 2031, 2042, 2488, 1012, 2023, 3185, 3319, 2003,
          102,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0]])
tensor([[  101,  2023,  3185,  2001,  1037,  5949,  1997,  2026,  2051,  1010,
          1045,  2428, 18966,  2673,  2055,  2009,  1012,  2023,  3185,  3319,
          2003,   102,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[ 101, 4361, 2003, 1037, 2103, 1999, 4561, 1010, 2710, 1012, 2023, 3185,
         3319, 2003,  102,    0,    0,    0,    0,    0,    0,    0,    0,    0,
            0,    0,    0,    0,    0,    0]])
tensor([[  101,  1045,  2428,  3866,  2023,  3185,   999,  1996,  3772,  2001,
          6887, 16515, 22311,  2140,  1998,  1996, 16434,  2001,  6429,  1012,
          2023, 11508,  3012,  1997,  1996,  6251,  2003,   102,     0,     0]])
tensor([[  101,  2023,  3185,  2001, 10303,  1010,  1045,  2074,  4669,  1996,
          4566,  1012,  2023, 11508,  3012,  1997,  1996,  6251,  2003,   102,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[  101,  1996,  3772,  2071,  2031,  2042,  2488,  1012,  2023, 11508,
          3012,  1997,  1996,  6251,  2003,   102,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[  101,  2023,  3185,  2001,  1037,  5949,  1997,  2026,  2051,  1010,
          1045,  2428, 18966,  2673,  2055,  2009,  1012,  2023, 11508,  3012,
```

```
           1997,  1996,  6251,  2003,   102,     0,     0,     0,     0,     0]])
tensor([[  101,  4361,  2003,  1037,  2103,  1999,  4561,  1010,  2710,  1012,
          2023, 11508,  3012,  1997,  1996,  6251,  2003,   102,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[  101,  1045,  2428,  3866,  2023,  3185,   999,  1996,  3772,  2001,
          6887, 16515, 22311,  2140,  1998,  1996, 16434,  2001,  6429,  1012,
          2023, 11508,  3012,  1997,  1996,  6251,  2003,   102,     0,     0]])
tensor([[  101,  2023,  3185,  2001, 10303,  1010,  1045,  2074,  4669,  1996,
          4566,  1012,  2023, 11508,  3012,  1997,  1996,  6251,  2003,   102,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[  101,  1996,  3772,  2071,  2031,  2042,  2488,  1012,  2023, 11508,
          3012,  1997,  1996,  6251,  2003,   102,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
tensor([[  101,  2023,  3185,  2001,  1037,  5949,  1997,  2026,  2051,  1010,
          1045,  2428, 18966,  2673,  2055,  2009,  1012,  2023, 11508,  3012,
          1997,  1996,  6251,  2003,   102,     0,     0,     0,     0,     0]])
tensor([[  101,  4361,  2003,  1037,  2103,  1999,  4561,  1010,  2710,  1012,
          2023, 11508,  3012,  1997,  1996,  6251,  2003,   102,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0]])
```

In [ ]:

```python
outputs = []
yes_id = tokenizer.convert_tokens_to_ids('yes')
no_id = tokenizer.convert_tokens_to_ids('no')
for i in range(5):
    print("-----= Sentence " + str(i+1) + " ==----")
    outputs.append(modelLM.forward(input_ids=id_list[i],attention_mask=attention_ma
    outputs[i].keys()

    #Finds first index where sentence ends
    end_index = torch.where(id_list[i][0]==0)[0][0].item()

    token_probabilities = F.softmax(outputs[i]['logits'], dim=-1)

    #Normalize probabilities of yes/no
    yes_prob = token_probabilities[0, end_index, yes_id].detach()
    no_prob = token_probabilities[0, end_index, no_id].detach()
    print(yes_prob / (yes_prob + no_prob),"yes")
    print(no_prob / (yes_prob + no_prob),"no")
```

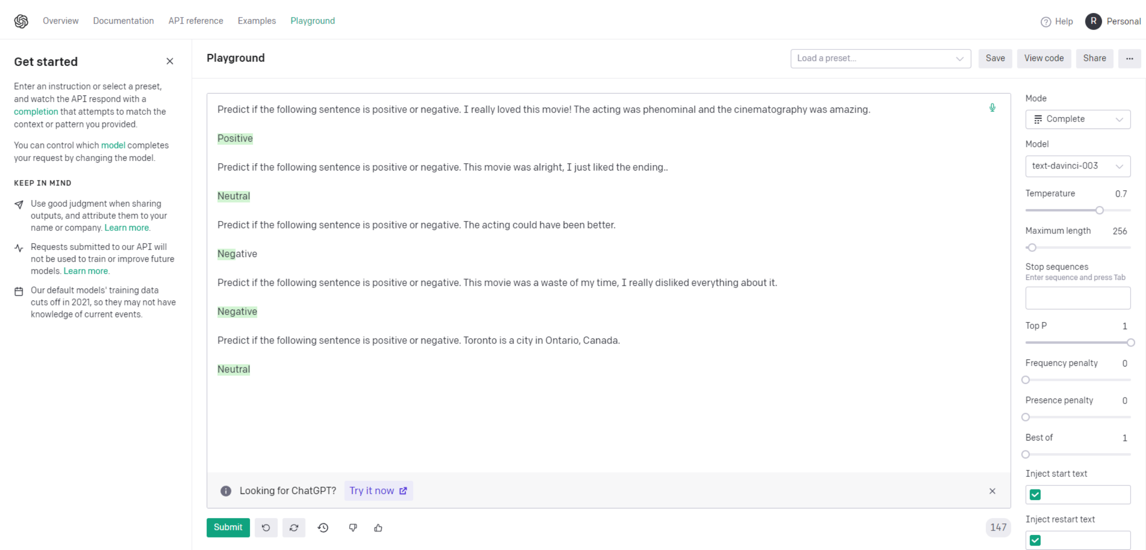
```
--=== Sentence 1 ===---
tensor(0.2271) yes
tensor(0.7729) no
--=== Sentence 2 ===---
tensor(0.4832) yes
tensor(0.5168) no
--=== Sentence 3 ===---
tensor(0.2220) yes
tensor(0.7780) no
--=== Sentence 4 ===---
tensor(0.3434) yes
tensor(0.6566) no
--=== Sentence 5 ===---
tensor(0.1568) yes
tensor(0.8432) no
```

Comments

This model performed very poorly compared to the previous one, it believes that all the movie reviews are negative when only 2/4 are explicitly negative. Even modifying the "This movie review is" addition to the sentences didn't help either, it even somehow made the random sentence the most polarizing of the bunch, with a probability of 84% being a negative review over a positive one. The previous model was definitely more successful than this model.

4.3

Using GPT3's playground, I was able to output the following predictions using a format of "Predict if the following sentence is positive or negative. + Sentence". A temperature hyper-parameter setting of 0.7 was optimal as it was able to successfully classify each sentence. The only issue was that the slightly positive sentence was categorized as neutral, but even that can be up to interpretation for humans.



Comments

Overall this was the best performing model out of 4.1 and 4.2, as it was also able to determine how neutral the sentence was too.