StoryGenerator

December 1, 2021

[ ]:
```
import torch
import pandas as pd
import numpy as np
```

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

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

[ ]:
```
#need to load data
df = pd.read_csv("/content/drive/MyDrive/Colab Notebooks/ROCStories_winter2017␣
↪- ROCStories_winter2017.csv")
```

[ ]:
```
pd.set_option('display.max_columns', None)
print(df.head())
```

```
                                storyid               storytitle  \
0  8bbe6d11-1e2e-413c-bf81-eaea05f4f1bd   David Drops the Weight
1  0beabab2-fb49-460e-a6e6-f35a202e3348              Frustration
2  87da1a22-df0b-410c-b186-439700b70ba6       Marcus Buys Khakis
3  2d16bcd6-692a-4fc0-8e7c-4a6f81d9efa9       Different Opinions
4  c71bb23b-7731-4233-8298-76ba6886cee1  Overcoming shortcomings

                                        sentence1  \
0  David noticed he had put on a lot of weight re…
1                      Tom had a very short temper.
2  Marcus needed clothing for a business casual e…
3  Bobby thought Bill should buy a trailer and ha…
4         John was a pastor with a very bad memory.

                                        sentence2  \
0  He examined his habits to try and figure out t…
1              One day a guest made him very angry.
2  All of his clothes were either too formal or t…
3  Bill thought a truck would be better for what …
4  He tried to memorize his sermons many days in …

                                        sentence3  \
0  He realized he'd been eating too much fast foo…
```

```
1        He punched a hole in the wall of his house.
2                He decided to buy a pair of khakis.
3  Bobby pointed out two vehicles were much more …
4  He decided to learn to sing to overcome his ha…

                                              sentence4  \
0  He stopped going to burger places and started …
1        Tom's guest became afraid and left quickly.
2              The pair he bought fit him perfectly.
3  Bill was set in his ways with conventional thi…
4  He then made all his sermons into music and sa…

                                              sentence5
0  After a few weeks, he started to feel much bet…
1  Tom sat on his couch filled with regret about …
2  Marcus was happy to have the right clothes for…
3  He ended up buying the truck he wanted despite…
4      His congregation was delighted and so was he.
```

0.1 Storyline Planning (static)

Use BiLSTM for encoding and LSTM for decoding

[ ]:
```
!pip install rake-nltk
from rake_nltk import Rake
import nltk

nltk.download('stopwords')
nltk.download('punkt')

from nltk.corpus import stopwords

# Uses stopwords for english from NLTK, and all punctuation characters by
# default
#max_length=1 so that we only get one word
r = Rake(max_length=1, min_length=1, include_repeated_phrases=False)

def rake_implement(s1, s2, s3, s4, s5, r):
    result = []
    result.append(get_word(r, s1))
    result.append(get_word(r, s2))
    result.append(get_word(r, s3))
    result.append(get_word(r, s4))
    result.append(get_word(r, s5))
    return result

def get_word(r, s):
  r.extract_keywords_from_text(s)
```

```
  phrases = r.get_ranked_phrases()
  if(len(phrases) == 0):
    #if nothing is extracted, take first word that is not in stopwords (from␣
 ↪the middle)
    sentence = s.split()
    for i in range(int(len(sentence)/2), len(sentence)):
      if not sentence[i] in stopwords.words('english'):
        return sentence[i].lower()
  else:
    return phrases[0]

#https://stackoverflow.com/questions/56836477/
 ↪apply-nltk-rake-to-each-row-in-dataframe
df['train_storylines'] = df.apply(lambda x: rake_implement(x.sentence1, x.
 ↪sentence2, x.sentence3, x.sentence4, x.sentence5, r), axis=1)

print(df.head())
```

```
Collecting rake-nltk
  Downloading rake_nltk-1.0.6-py3-none-any.whl (9.1 kB)
Collecting nltk<4.0.0,>=3.6.2
  Downloading nltk-3.6.5-py3-none-any.whl (1.5 MB)
     |                  | 1.5 MB 5.3 MB/s
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-
packages (from nltk<4.0.0,>=3.6.2->rake-nltk) (1.1.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/dist-packages
(from nltk<4.0.0,>=3.6.2->rake-nltk) (4.62.3)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages
(from nltk<4.0.0,>=3.6.2->rake-nltk) (7.1.2)
Collecting regex>=2021.8.3
  Downloading
regex-2021.11.10-cp37-cp37m-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (749
kB)
     |                  | 749 kB 21.1 MB/s
Installing collected packages: regex, nltk, rake-nltk
  Attempting uninstall: regex
    Found existing installation: regex 2019.12.20
    Uninstalling regex-2019.12.20:
      Successfully uninstalled regex-2019.12.20
  Attempting uninstall: nltk
    Found existing installation: nltk 3.2.5
    Uninstalling nltk-3.2.5:
      Successfully uninstalled nltk-3.2.5
Successfully installed nltk-3.6.5 rake-nltk-1.0.6 regex-2021.11.10

[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package punkt to /root/nltk_data…
```

```
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

	storyid	storytitle	sentence1	sentence2	sentence3	sentence4	sentence5	train_storylines
0	8bbe6d11-1e2e-413c-bf81-eaea05f4f1bd	David Drops the Weight	David noticed he had put on a lot of weight re…	He examined his habits to try and figure out t…	He realized he'd been eating too much fast foo…	He stopped going to burger places and started …	After a few weeks, he started to feel much bet…	[put, try, realized, started, weeks]
1	0beabab2-fb49-460e-a6e6-f35a202e3348	Frustration	Tom had a very short temper.	One day a guest made him very angry.	He punched a hole in the wall of his house.	Tom's guest became afraid and left quickly.	Tom sat on his couch filled with regret about …	[tom, angry, wall, tom, regret]
2	87da1a22-df0b-410c-b186-439700b70ba6	Marcus Buys Khakis	Marcus needed clothing for a business casual e…	All of his clothes were either too formal or t…	He decided to buy a pair of khakis.	The pair he bought fit him perfectly.	Marcus was happy to have the right clothes for…	[business, formal, pair, perfectly, marcus]
3	2d16bcd6-692a-4fc0-8e7c-4a6f81d9efa9	Different Opinions	Bobby thought Bill should buy a trailer and ha…	Bill thought a truck would be better for what …	Bobby pointed out two vehicles were much more …	Bill was set in his ways with conventional thi…	He ended up buying the truck he wanted despite…	
4	c71bb23b-7731-4233-8298-76ba6886cee1	Overcoming shortcomings	John was a pastor with a very bad memory.	He tried to memorize his sermons many days in …	He decided to learn to sing to overcome his ha…	He then made all his sermons into music and sa…	His congregation was delighted and so was he.	

```
3          [trailer, needed, much, ways, truck]
4     [pastor, tried, sing, sundays, delighted]
```

[ ]:
```python
#to convert words to indices and vice versa
#https://pytorch.org/tutorials/intermediate/seq2seq_translation_tutorial.html
class Vocab:
    def __init__(self, name):
        self.name = name
        self.word2index = {"<SOS>": 0, "<EOT>": 1, "<EOS>": 2}
        self.word2count = {}
        self.index2word = {0: "<SOS>", 1:"<EOT>", 2:"<EOS>"}
        self.n_words = 3

    def addTitle(self, title):
      for word in title.split():
        self.addWordToCount(word)

    def addStoryline(self, storyline):
        for word in storyline:
            #could do preprocessing here
            self.addWordToCount(word)

    def addSentence(self, sentence):
      for word in sentence.split():
        self.addWordToCount(word)

    def addWordToCount(self, word):
        if word not in self.word2count:
            self.word2count[word] = 1
        else:
            self.word2count[word] += 1

    def addWordToDicts(self, word):
      self.word2index[word] = self.n_words
      self.index2word[self.n_words] = word
      self.n_words += 1

    def removeWordFromCount(self, word):
      self.word2count.pop(word)

    def length(self):
      return len(self.index2word)
```

[ ]:
```python
def prepareVocab(df):
  vocab = Vocab("trainingVocab")
  for title in df["storytitle"]:
    vocab.addTitle(title)
```

```
  for s in df["sentence1"]:
    vocab.addSentence(s)
  for s in df["sentence2"]:
    vocab.addSentence(s)
  for s in df["sentence3"]:
    vocab.addSentence(s)
  for s in df["sentence4"]:
    vocab.addSentence(s)
  for s in df["sentence5"]:
    vocab.addSentence(s)
  for storyline in df["train_storylines"]:
    vocab.addStoryline(storyline)

  #go thru counts and combine all those with just 1 count into <unk>
  #unkCount = 0
  #low_dict = {key: value for key, value in vocab.word2count.items() if value
 ↪== 1}
  #for key, value in low_dict.items():
    #add 1 to unkCount and remove from dict
    #unkCount += 1
    #vocab.removeWordFromCount(key)
  # add unk, then add rest of words to dicts
  #vocab.addWordToDicts("<unk>")
  for key, value in vocab.word2count.items():
    vocab.addWordToDicts(key)

  return vocab
```

[ ]:
```
# divide into training, (dev and test)
msk = np.random.rand(len(df)) < 0.01 #reducing training just to test something
 ↪out (0.8)
reduced_df = df[msk]
msk = np.random.rand(len(reduced_df)) < 0.8
train_df = reduced_df[msk]
eval_df = reduced_df[~msk]
msk = np.random.rand(len(eval_df)) < 0.5
dev_df = eval_df[msk]
test_df = eval_df[~msk]
```

[ ]:
```
#prepare vocab! Put both training and test?
vocab = prepareVocab(reduced_df)
```

[ ]:
```
INPUT_SIZE = 15
HIDDEN_SIZE = 1000
EMBEDDING_SIZE = 500
OUTPUT_SIZE = vocab.length()
TARGET_SIZE = 5
```

```
NUM_LAYERS = 1
BATCH_SIZE = 50
NUM_WORKERS = 0
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

[ ]:
```
# create datasets
from torch.utils.data import DataLoader, Dataset
class StorylineDataset (Dataset):
    def __init__(self, titles, storylines, vocab, input_size):
        #must convert words to numeric representations
        self.titles = []
        for title in titles:
          temp = []
          for word in title.split():
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
          #pad
          self.titles.append(np.pad(temp, (input_size-len(temp), 0),␣
 ↪'constant'))

        self.storylines = []
        for storyline in storylines:
          temp = []
          for word in storyline:
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
          self.storylines.append(temp)

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

    def __getitem__(self, index):
        title = np.asarray(self.titles[index])
        storyline = np.asarray(self.storylines[index])

        title = torch.from_numpy(title).long()
        storyline = torch.from_numpy(storyline).long()
        return (title, storyline)
```

[ ]:
```
training_data = StorylineDataset(train_df["storytitle"],␣
 ↪train_df["train_storylines"], vocab, INPUT_SIZE)
testing_data = StorylineDataset(test_df["storytitle"],␣
 ↪test_df["train_storylines"], vocab, INPUT_SIZE)
```

[ ]:
```
training_loader = torch.utils.data.DataLoader(training_data,␣
↪batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)
testing_loader = torch.utils.data.DataLoader(testing_data,␣
↪batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)
```

[ ]:
```
#encoder
class TitleEncoder(torch.nn.Module):
    def __init__(self, input_size, hidden_size, embedding_size, vocab_size,␣
↪num_layers):
        super(TitleEncoder, self).__init__()
        self.hidden_size = hidden_size
        self.input_size = input_size
        self.embedding_size = embedding_size
        self.vocab_size = vocab_size

        #print(vocab_size)
        #print(embedding_size)
        self.embedding = torch.nn.Embedding(vocab_size, embedding_size)
        self.bilstm = torch.nn.LSTM(self.embedding_size, self.hidden_size,␣
↪num_layers=num_layers, batch_first=True, bidirectional=True)
        self.hidden2hidden = torch.nn.Linear(2*self.hidden_size, self.
↪hidden_size)

    def forward(self, title):
        embedded = self.embedding(title)
        bilstm_out, hidden_out = self.bilstm(embedded)
        #print(hidden_out)
        #print(type(hidden_out))
        output = self.hidden2hidden(bilstm_out)
        return output, hidden_out
```

[ ]:
```
#decoder
# need to incorporate attention: https://pytorch.org/tutorials/intermediate/
↪seq2seq_translation_tutorial.html
#initial input token is the start of string token, and the first hidden state␣
↪si the context vector
class StorylineDecoder(torch.nn.Module):
    def __init__(self, hidden_size, output_size, embedding_size, num_layers,␣
↪e_dropout_p, h_dropout_p=0.1):
        super(StorylineDecoder, self).__init__()
        self.hidden_size = hidden_size
        self.embedding_size = embedding_size
        self.output_size = output_size
        self.e_dropout_p = e_dropout_p
        self.h_dropout_p = h_dropout_p
```

```
        self.embedding = torch.nn.Embedding(self.output_size, self.
↪embedding_size)
        self.lstm = torch.nn.LSTM(self.embedding_size, self.hidden_size,␣
↪num_layers=num_layers, batch_first=True)
        self.out = torch.nn.Linear(self.hidden_size, self.output_size)
        #self.softmax = torch.nn.LogSoftmax(dim=1)
        self.e_dropout = torch.nn.Dropout(self.e_dropout_p)
        self.h_dropout = torch.nn.Dropout(self.h_dropout_p)

    def forward(self, input, hidden):
        embedded = self.embedding(input)
        embedded = self.e_dropout(embedded)
        lstm_out, hidden_out = self.lstm(embedded, hidden)
        output = self.out(lstm_out)
        output = self.h_dropout(output)
        #output = self.softmax(output)
        return output, hidden_out
```

[ ]:
```
def train(input_tensor, target_tensor, encoder, decoder, encoder_optimizer,␣
↪decoder_optimizer, criterion, batch_size=BATCH_SIZE,␣
↪target_length=TARGET_SIZE):
    encoder_hidden = torch.zeros(1, 1, HIDDEN_SIZE, device=device)

    encoder_optimizer.zero_grad()
    decoder_optimizer.zero_grad()

    input_length = INPUT_SIZE#input_tensor.size(0)
    #target_length = TARGET_SIZE#target_tensor.size(0)

    #encoder_outputs = torch.zeros(max_length, encoder.hidden_size,␣
↪device=device)

    loss = 0

    #batch first
    encoder_output, hidden = encoder(input_tensor)
    #encoder_outputs[ei] = encoder_output[0, 0]

    #start with SOS token
    decoder_input = torch.tensor([[0]], device=device)


    use_teacher_forcing = False #True if random.random() <␣
↪teacher_forcing_ratio else False

    if use_teacher_forcing:
        # Teacher forcing: Feed the target as the next input
```

```
        for di in range(target_length):
            decoder_output, decoder_hidden, decoder_attention = decoder(
                decoder_input, decoder_hidden, encoder_outputs)
            loss += criterion(decoder_output, target_tensor[di])
            decoder_input = target_tensor[di]  # Teacher forcing

    else:
        # Without teacher forcing: use its own predictions as the next input
        #go thru batch first
        for b in range(len(hidden[0][0])):
          h_0 = torch.zeros(1, 1, HIDDEN_SIZE*2, device=device)
          h_0[0][0] = hidden[0].transpose(0,1)[b].flatten()
          c_0 = torch.zeros(1, 1, HIDDEN_SIZE*2, device=device)
          c_0[0][0] = hidden[1].transpose(0,1)[b].flatten()
          decoder_hidden = (h_0, c_0)

          for di in range(target_length):
              decoder_output, decoder_hidden = decoder(decoder_input,␣
↪decoder_hidden)
              topv, topi = decoder_output.topk(1)
              decoder_input = topi.squeeze().detach()  # detach from history as␣
↪input
              decoder_input = torch.tensor([[decoder_input]], device=device)
              #print(target_tensor)
              reshaped_target = torch.zeros(1, device=device).long()
              reshaped_target[0] = target_tensor[b][di]
              #print(target_tensor[b][di])
              #print(reshaped_target)
              #print(vocab.index2word[topi[0][0][0].item()])
              #print(vocab.index2word[reshaped_target[0].item()])
              #print(decoder_output)
              #print(decoder_output[0])
              loss += criterion(decoder_output[0], reshaped_target)

    #print("backwards step")
    loss.backward()
    print("loss: ", loss.item() / (len(hidden[0][0])*target_length))

    encoder_optimizer.step()
    decoder_optimizer.step()

    return loss.item() / (len(hidden[0][0])*target_length)
```

[ ]:
```
#init models
encoder_model = TitleEncoder(INPUT_SIZE, HIDDEN_SIZE, EMBEDDING_SIZE, vocab.
↪length(), NUM_LAYERS).to(device)
```

```
decoder_model = StorylineDecoder(HIDDEN_SIZE*2, OUTPUT_SIZE, EMBEDDING_SIZE,␣
↪NUM_LAYERS, 0.4).to(device)
```

[ ]:
```
#load encoder_model
encoder_model.load_state_dict(torch.load("/content/drive/MyDrive/Colab␣
↪Notebooks/encoder_title.pt"))
```

[ ]:
```
#load decoder_model
decoder_model.load_state_dict(torch.load("/content/drive/MyDrive/Colab␣
↪Notebooks/decoder_storyline.pt"))
```

[ ]:
```
# create criterion and optimizer
criterion = torch.nn.CrossEntropyLoss()
encoder_optimizer = torch.optim.SGD(encoder_model.parameters(), lr = 0.01,␣
↪momentum=0.9)#torch.optim.Adam(encoder_model.parameters(), lr=0.01) #torch.
↪optim.SGD(encoder_model.parameters(), lr = 0.01, momentum=0.9)
decoder_optimizer = torch.optim.SGD(decoder_model.parameters(), lr = 0.01,␣
↪momentum=0.9)#torch.optim.Adam(decoder_model.parameters(), lr=0.01) #torch.
↪optim.SGD(decoder_model.parameters(), lr = 0.01, momentum=0.9)
#scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.9)
```

[ ]:
```
def trainModel(num_epochs, train_loader, test_loader, encoder_model,␣
↪decoder_model, encoder_optimizer, decoder_optimizer, criterion):
  #keep track of validation loss
  valid_loss_min = float('inf')
  for epoch in range(1, num_epochs+1):
            total_loss = 0.0
            #put models in train mode
            encoder_model.train()
            decoder_model.train()
            for titles, storylines in train_loader:
              #send to cuda
              titles.to(device)
              storylines.to(device)
              #call train function
              total_loss += train(titles, storylines, encoder_model,␣
↪decoder_model, encoder_optimizer, decoder_optimizer, criterion)

            print("Epoch ", epoch, " training loss: ", total_loss/
↪len(train_loader))
```

[ ]:
```
# call trainModel 0.047
trainModel(3, training_loader, testing_loader, encoder_model, decoder_model,␣
↪encoder_optimizer, decoder_optimizer, criterion)
```

```
loss:  20.149923828125
loss:  24.896830078125
loss:  17.19998828125
```

```
loss:  25.257783203125
loss:  25.016263671875
loss:  24.500158203125
loss:  20.377439453125
loss:  27.589599609375
loss:  19.265682547433034
Epoch  1  training loss:  22.694852097284226
loss:  17.20325390625
loss:  19.303689453125
loss:  16.496734375
loss:  19.777203125
loss:  18.242998046875
loss:  25.30678125
loss:  18.636916015625
loss:  20.6531484375
loss:  21.169011579241072
Epoch  2  training loss:  19.64330402095734
loss:  17.0391640625
loss:  16.0817314453125
loss:  12.9033359375
loss:  15.1892333984375
loss:  16.36453515625
loss:  14.4716396484375
loss:  16.3831845703125
loss:  13.1266572265625
loss:  12.277472795758928
Epoch  3  training loss:  14.87077269345238
```

[ ]:
```
#save encoder_model
torch.save(encoder_model.state_dict(), "/content/drive/MyDrive/Colab Notebooks/
 ↪encoder_title.pt")
```

[ ]:
```
#save decoder_model
torch.save(encoder_model.state_dict(), "/content/drive/MyDrive/Colab Notebooks/
 ↪decoder_storyline.pt")
```

[ ]:
```
def evaluate(input_tensor, target_tensor, encoder, decoder, criterion,␣
 ↪batch_size=BATCH_SIZE, target_length=TARGET_SIZE):
  with torch.no_grad():
    encoder_hidden = torch.zeros(1, 1, HIDDEN_SIZE, device=device)

    input_length = INPUT_SIZE#input_tensor.size(0)
    #target_length = TARGET_SIZE#target_tensor.size(0)

    loss = 0

    #batch first
```

```
    encoder_output, hidden = encoder(input_tensor)
    #encoder_outputs[ei] = encoder_output[0, 0]

    #start with SOS token
    decoder_input = torch.tensor([[0]], device=device)

    titles = []
    predicteds = []
    targets = []


    # Without teacher forcing: use its own predictions as the next input
    #go thru batch first
    for b in range(len(hidden[0][0])):
      title = []
      predicted = []
      target = []
      for i in range(len(input_tensor[b])):
        title.append(vocab.index2word[input_tensor[b][i].item()])

      h_0 = torch.zeros(1, 1, HIDDEN_SIZE*2, device=device)
      h_0[0][0] = hidden[0].transpose(0,1)[b].flatten()
      c_0 = torch.zeros(1, 1, HIDDEN_SIZE*2, device=device)
      c_0[0][0] = hidden[1].transpose(0,1)[b].flatten()
      decoder_hidden = (h_0, c_0)

      for di in range(target_length):
          decoder_output, decoder_hidden = decoder(decoder_input,␣
↪decoder_hidden)
          topv, topi = decoder_output.topk(1)
          decoder_input = topi.squeeze().detach()  # detach from history as␣
↪input
          decoder_input = torch.tensor([[decoder_input]], device=device)
          #print(target_tensor)
          reshaped_target = torch.zeros(1, device=device).long()
          reshaped_target[0] = target_tensor[b][di]
          loss += criterion(decoder_output[0], reshaped_target)

          predicted.append(vocab.index2word[topi[0][0][0].item()])
          target.append(vocab.index2word[reshaped_target[0].item()])

      if (b == 0 or b == len(hidden[0][0])-1):
        print("title: ", title)
        print("predicted: ", predicted)
        print("Target: ", target)
        print('\n')
      #append to output lists
```

```
      titles.append(title)
      predicteds.append(predicted)
      targets.append(target)

    return (loss.item() / (len(hidden[0][0])*target_length), titles,␣
↪predicteds, targets)
```

[ ]:
```
def evaluateModel(test_loader, encoder_model, decoder_model, criterion):
  #keep track of validation loss
  valid_loss_min = float('inf')
  total_loss = 0.0
  for titles, storylines in test_loader:
    #send to cuda
    titles.to(device)
    storylines.to(device)
    #call train function
    loss, titles, predicteds, targets = evaluate(titles, storylines,␣
↪encoder_model, decoder_model, criterion)
    total_loss += loss
  print("total loss: ", total_loss/len(test_loader))
```

[ ]:
```
evaluateModel(testing_loader, encoder_model, decoder_model, criterion)
```

```
title:  ['<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', 'Mary', "isn't", 'home']
predicted:  ['party', 'wanted', 'home', 'party', 'car']
Target:  ['mary', 'mall', 'running', 'pulled', 'knocks']


title:  ['<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', 'No', 'Help']
predicted:  ['party', 'car', 'go', 'car', 'unfortunately']
Target:  ['solving', 'help', 'wanted', 'hour', 'solved']


title:  ['<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', 'The', 'Long', 'Jump']
predicted:  ['party', 'car', 'go', 'car', 'unfortunately']
Target:  ['jose', 'ready', 'began', 'track', 'crashed']


title:  ['<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', 'The', 'Wrong', 'Brand']
predicted:  ['party', 'go', 'car', 'unfortunately', 'unfortunately']
Target:  ['nate', 'buying', 'upset', 'name', 'nate']


total loss:  14.117823079427083
```

[ ]:
```
# create datasets
from torch.utils.data import DataLoader, Dataset
class SentencesDataset (Dataset):
    def __init__(self, titles, storylines, sentence1, sentence2, sentence3,␣
 ↪sentence4, sentence5, vocab, input_size, output_size):
        #must convert words to numeric representations
        #convert all into separate arrays first
        self.titles = []
        for title in titles:
          temp = []
          for word in title.split():
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
          #pad
          self.titles.append(temp)

        self.storylines = []
        for storyline in storylines:
          temp = []
          for word in storyline:
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
          self.storylines.append(temp)

        #convert titles and storylines to combined_input
        self.combined_input = []
        for i in range(len(self.titles)):
          temp = []
          temp.extend(self.titles[i])
          temp.append(vocab.word2index['<EOT>'])
          temp.extend(self.storylines[i])
          #pad to input_size amount
          self.combined_input.append(np.pad(temp, (input_size-len(temp), 0),␣
 ↪'constant'))

        self.sentences = []
        for i in range(len(sentence1)):
          temp = []
          for word in sentence1.iloc[i].split():
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
```

```
          temp.append(vocab.word2index["<EOS>"])
          for word in sentence2.iloc[i].split():
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
          temp.append(vocab.word2index["<EOS>"])
          for word in sentence3.iloc[i].split():
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
          temp.append(vocab.word2index["<EOS>"])
          for word in sentence4.iloc[i].split():
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
          temp.append(vocab.word2index["<EOS>"])
          for word in sentence5.iloc[i].split():
            if word in vocab.word2index:
              temp.append(vocab.word2index[word])
            else:
              temp.append(vocab.word2index[""])
          temp.append(vocab.word2index["<EOS>"])
          #append full story to sentences, making sure to pad
          self.sentences.append(np.pad(temp, (0, output_size-len(temp)),␣
↪'constant'))

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

    def __getitem__(self, index):
        title = np.asarray(self.titles[index])
        storyline = np.asarray(self.storylines[index])
        input = np.asarray(self.combined_input[index])
        story = np.asarray(self.sentences[index])

        #title = torch.from_numpy(title).long()
        #storyline = torch.from_numpy(storyline).long()
        input = torch.from_numpy(input).long()
        story = torch.from_numpy(story).long()
        return (input, story)
```

[ ]:
```
COMBINED_INPUT_SIZE = 20
STORY_OUTPUT_SIZE = 100
```

```
[ ]: training_data_story = SentencesDataset(train_df["storytitle"],␣
     ↪train_df["train_storylines"], train_df["sentence1"], train_df["sentence2"],␣
     ↪train_df["sentence3"], train_df["sentence4"], train_df["sentence5"], vocab,␣
     ↪COMBINED_INPUT_SIZE, STORY_OUTPUT_SIZE)
     testing_data_story = SentencesDataset(test_df["storytitle"],␣
     ↪test_df["train_storylines"], test_df["sentence1"], test_df["sentence2"],␣
     ↪test_df["sentence3"], test_df["sentence4"], test_df["sentence5"], vocab,␣
     ↪COMBINED_INPUT_SIZE, STORY_OUTPUT_SIZE)
```

```
[ ]: training_loader_story = torch.utils.data.DataLoader(training_data_story,␣
     ↪batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)
     testing_loader_story = torch.utils.data.DataLoader(testing_data_story,␣
     ↪batch_size=BATCH_SIZE, num_workers=NUM_WORKERS)
```

```
[ ]: #can we use previous model functions with different parameters?
     #init models
     encoder_model_story = TitleEncoder(COMBINED_INPUT_SIZE, HIDDEN_SIZE,␣
     ↪EMBEDDING_SIZE, vocab.length(), NUM_LAYERS).to(device)
     decoder_model_story = StorylineDecoder(HIDDEN_SIZE*2, OUTPUT_SIZE,␣
     ↪EMBEDDING_SIZE, NUM_LAYERS, 0.2).to(device)
```

```
[ ]: # create criterion and optimizer
     criterion_story = torch.nn.CrossEntropyLoss() #torch.nn.NLLLoss()
     encoder_optimizer_story = torch.optim.SGD(encoder_model.parameters(), lr = 0.
     ↪01, momentum=0.9)
     decoder_optimizer_story = torch.optim.SGD(decoder_model.parameters(), lr = 0.
     ↪01, momentum=0.9)
     #scheduler = torch.optim.lr_scheduler.StepLR(optimizer, step_size=30, gamma=0.9)
```

```
[ ]: def trainModelStory(num_epochs, train_loader, test_loader, encoder_model,␣
     ↪decoder_model, encoder_optimizer, decoder_optimizer, criterion):
       #keep track of validation loss
       valid_loss_min = float('inf')
       for epoch in range(1, num_epochs+1):
                 total_loss = 0.0
                 #put models in train mode
                 encoder_model.train()
                 decoder_model.train()
                 for inputs, stories in train_loader:
                   #send to cuda
                   inputs.to(device)
                   stories.to(device)
                   #call train function
                   total_loss += train(inputs, stories, encoder_model,␣
     ↪decoder_model, encoder_optimizer, decoder_optimizer, criterion,␣
     ↪target_length=STORY_OUTPUT_SIZE)
                   #break #for testing purposes only
```

```
            print("Epoch ", epoch, " training loss: ", total_loss/
 ↪len(train_loader))
```

[ ]:
```
# call trainModel
print(vocab.length())
trainModelStory(1, training_loader_story, testing_loader_story,␣
 ↪encoder_model_story, decoder_model_story, encoder_optimizer_story,␣
 ↪decoder_optimizer_story, criterion_story)
```

```
5643
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-56-0c4ef446c22e> in <module>()
      1 # call trainModel
      2 print(vocab.length())
----> 3 trainModelStory(1, training_loader_story, testing_loader_story,␣
 ↪encoder_model_story, decoder_model_story, encoder_optimizer_story,␣
 ↪decoder_optimizer_story, criterion_story)

<ipython-input-55-674afced3e4e> in trainModelStory(num_epochs, train_loader,␣
 ↪test_loader, encoder_model, decoder_model, encoder_optimizer,␣
 ↪decoder_optimizer, criterion)
     12                 stories.to(device)
     13                 #call train function
---> 14                 total_loss += train(inputs, stories, encoder_model,␣
 ↪decoder_model, encoder_optimizer, decoder_optimizer, criterion,␣
 ↪target_length=STORY_OUTPUT_SIZE)
     15                 #break #for testing purposes only
     16             print("Epoch ", epoch, " training loss: ", total_loss/
 ↪len(train_loader))

<ipython-input-36-266d1cff4a99> in train(input_tensor, target_tensor, encoder,␣
 ↪decoder, encoder_optimizer, decoder_optimizer, criterion, batch_size,␣
 ↪target_length)
     57
     58     #print("backwards step")
---> 59     loss.backward()
     60     print("loss: ", loss.item() / (len(hidden[0][0])*target_length))
     61

/usr/local/lib/python3.7/dist-packages/torch/_tensor.py in backward(self,␣
 ↪gradient, retain_graph, create_graph, inputs)
    305                 create_graph=create_graph,
    306                 inputs=inputs)
--> 307         torch.autograd.backward(self, gradient, retain_graph,␣
 ↪create_graph, inputs=inputs)
    308
    309     def register_hook(self, hook):
```

```
/usr/local/lib/python3.7/dist-packages/torch/autograd/__init__.py in␣
 ↪backward(tensors, grad_tensors, retain_graph, create_graph, grad_variables,␣
 ↪inputs)
    154     Variable._execution_engine.run_backward(
    155         tensors, grad_tensors_, retain_graph, create_graph, inputs,
--> 156         allow_unreachable=True, accumulate_grad=True)  #␣
 ↪allow_unreachable flag
    157
    158

KeyboardInterrupt:
```

[ ]:
```
#save encoder_model
torch.save(encoder_model_story.state_dict(), "/content/drive/MyDrive/Colab␣
 ↪Notebooks/encoder_combined.pt")
```

[ ]:
```
#save decoder_model
torch.save(decoder_model_story.state_dict(), "/content/drive/MyDrive/Colab␣
 ↪Notebooks/decoder_story.pt")
```

[ ]:
```
def evaluateModelStory(test_loader, encoder_model, decoder_model, criterion):
  #keep track of validation loss
  valid_loss_min = float('inf')
  total_loss = 0.0
  for titles, storylines in test_loader:
    #send to cuda
    titles.to(device)
    storylines.to(device)
    #call train function
    loss, titles, predicteds, targets = evaluate(titles, storylines,␣
 ↪encoder_model, decoder_model, criterion, target_length=STORY_OUTPUT_SIZE)
    total_loss += loss
  print("total loss: ", total_loss/len(test_loader))
```

[ ]:
```
evaluateModelStory(testing_loader_story, encoder_model_story,␣
 ↪decoder_model_story, criterion_story)
```

```
title:  ['<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', 'The', 'Solo', '<EOT>', 'solo', 'blew',
'time', 'gave', 'took']
predicted:  ['divorce', 'house!', 'house!', 'push.', 'negotiating', 'wetsuit.',
'bleachers.', "world's", "billy's", 'Margaret', 'following', 'application',
'old.', 'Tally', "He's", 'ecology', 'Whopper', 'wall,', 'theater.', 'test,',
'stomach', 'Thieving', 'water', 'sports', 'argument.', 'like', 'rather',
'rather', 'stock', 'horseshoe', 'should', 'removed,', 'fussed', 'time',
'glass.', 'names', 'names', 'cheating', 'names', 'names', 'cheating', 'emailed',
```

```
'numbers', 'restaurant.', 'hoop.', 'hoop.', 'Afterwards,', 'aimlessly', 'deb',
'knit', 'he', 'Student', 'sad.', 'Tea', 'countless', 'declined.', "Year's",
'more', 'before', 'spicy.', 'steps', 'presented', 'too.', 'within', 'paint',
'gas', 'match', 'humvee', 'insisted', 'realtor.', 'yet', 'King.', 'lugged',
'parlor.', 'river,', 'tom', 'parents,', 'Table', 'jar', 'mini', 'Once',
'treating.', 'howling', 'Clothes', 'paycheck.', 'Sylvie', 'new.', 'Kirby',
'actually', 'plate', 'steps', 'tickets.', 'boat', 'hear', 'light.', 'Emptying',
'want', 'negotiating', 'begged', 'wetsuit.']
Target:  ['Anthony', 'auditioned', 'for', 'a', 'solo.', '<EOS>', 'He', 'blew',
'the', 'judges', 'away.', '<EOS>', 'When', 'it', 'was', 'time', 'to',
'perform,', 'Anthony', 'got', 'cold', 'feet.', '<EOS>', 'He', 'gave', 'himself',
'a', 'pep', 'talk.', '<EOS>', 'Finally,', 'he', 'took', 'the', 'stage', 'and',
'sang.', '<EOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>']


title:  ['<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', "Surf's", 'Up!', '<EOT>', 'went', 'wetsuit',
'water', 'waited', 'shore']
predicted:  ['mad,', 'arrested', 'cross-country', 'Whopper', 'cross-country',
'Whopper', 'wall,', 'theater.', 'test,', 'stomach', 'Thieving', 'water',
'sports', 'argument.', 'like', 'rather', 'rather', 'stock', 'horseshoe',
'should', 'removed,', 'fussed', 'time', 'glass.', 'names', 'names', 'cheating',
'names', 'names', 'cheating', 'emailed', 'numbers', 'restaurant.', 'hoop.',
'hoop.', 'Afterwards,', 'aimlessly', 'deb', 'knit', 'he', 'Student', 'sad.',
'Tea', 'countless', 'declined.', "Year's", 'more', 'before', 'spicy.', 'steps',
'presented', 'too.', 'within', 'paint', 'gas', 'match', 'humvee', 'insisted',
'realtor.', 'yet', 'King.', 'lugged', 'parlor.', 'river,', 'tom', 'parents,',
'Table', 'jar', 'mini', 'Once', 'treating.', 'howling', 'Clothes', 'paycheck.',
'Sylvie', 'new.', 'Kirby', 'actually', 'plate', 'steps', 'tickets.', 'boat',
'hear', 'light.', 'Emptying', 'want', 'negotiating', 'begged', 'wetsuit.',
'role', 'draft', 'role!', 'bite', 'ingredients', 'tyler', 'determined',
'parlor.', 'turn,', 'virus', 'kickboxer.']
Target:  ['Last', 'weekend', 'I', 'grabbed', 'my', 'surfboard', 'and', 'went',
'down', 'to', 'the', 'beach.', '<EOS>', 'It', 'was', 'really', 'cold', 'out',
'so', 'I', 'had', 'to', 'wear', 'my', 'wetsuit.', '<EOS>', 'When', 'I', 'got',
'into', 'the', 'water', 'I', 'realized', 'that', 'it', 'was', 'warmer', 'than',
'the', 'air.', '<EOS>', 'I', 'waited', 'a', 'long', 'time', 'for', 'the',
'perfect', 'wave.', '<EOS>', 'The', 'perfect', 'wave', 'came', 'towards', 'me',
'and', 'I', 'rode', 'it', 'happily', 'to', 'shore.', '<EOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
```

```
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>']


title:  ['<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', 'Blind', 'Date', '<EOT>', 'tim', 'blind',
'tim', 'went', 'long']
predicted:  ['principle', "Matt's", 'xbox', 'am', 'tasted', 'disabled',
'amazing', 'parlor.', 'trips', 'morning,', 'morning,', 'morning,', 'skate',
'Ota', 'stay', 'tricks', 'turned', 'coma.', 'stay', 'Alabama', 'quiet,',
"Year's", 'hungry.', 'countless', 'support', 'Much', 'quiet,', 'bumped', 'days',
'friend', 'too.', 'taking', 'Return', 'Tonight', 'changes', 'though,',
'excitedly', 'Whopper', 'cross-country', 'Whopper', 'wall,', 'theater.',
'test,', 'stomach', 'Thieving', 'water', 'sports', 'argument.', 'like',
'rather', 'rather', 'stock', 'horseshoe', 'should', 'removed,', 'fussed',
'time', 'glass.', 'names', 'names', 'cheating', 'names', 'names', 'cheating',
'emailed', 'numbers', 'restaurant.', 'hoop.', 'hoop.', 'Afterwards,',
'aimlessly', 'deb', 'knit', 'he', 'Student', 'sad.', 'Tea', 'countless',
'declined.', "Year's", 'more', 'before', 'spicy.', 'steps', 'presented', 'too.',
'within', 'paint', 'gas', 'match', 'humvee', 'insisted', 'realtor.', 'yet',
'King.', 'lugged', 'parlor.', 'river,', 'tom', 'parents,']
Target:  ['Tim', 'had', 'been', 'alone', 'for', 'a', 'while.', '<EOS>', 'His',
'friends', 'set', 'him', 'up', 'on', 'a', 'blind', 'date.', '<EOS>', 'Tim',
'was', 'nervous', 'and', 'awkward', 'throughout.', '<EOS>', 'The', 'date',
'went', 'horribly', 'wrong.', '<EOS>', 'Tim', 'remained', 'single', 'for', 'a',
'long', 'time', 'after.', '<EOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>']


title:  ['<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', 'amnesia', '<EOT>', 'remembering',
'belongings', 'car', 'find', 'things']
predicted:  ['Cream', 'playing.', 'release', 'Hard', 'spare.', 'sound.',
'banning', 'further.', 'monkey.', 'carton', 'carton', 'carton', 'carton',
'grabbed', "They'd", 'exam.', 'carton', 'carton', 'carton', 'carton', 'grabbed',
"They'd", 'exam.', 'carton', 'carton', 'carton', 'carton', 'grabbed', "They'd",
'exam.', 'carton', 'carton', 'carton', 'carton', 'grabbed', "They'd", 'exam.',
'carton', 'carton', 'carton', 'carton', 'grabbed', "They'd", 'exam.', 'carton',
'carton', 'carton', 'carton', 'grabbed', "They'd", 'exam.', 'carton', 'carton',
'carton', 'carton', 'grabbed', "They'd", 'exam.', 'carton', 'carton', 'carton',
'carton', 'grabbed', "They'd", 'exam.', 'carton', 'carton', 'carton', 'carton',
'grabbed', "They'd", 'exam.', 'carton', 'carton', 'carton', 'carton', 'grabbed',
```

```
"They'd", 'exam.', 'carton', 'carton', 'carton', 'carton', 'grabbed', "They'd",
'exam.', 'carton', 'carton', 'carton', 'carton', 'grabbed', "They'd", 'exam.',
'carton', 'carton', 'carton', 'carton', 'grabbed', "They'd", 'exam.']
Target:  ['John', 'woke', 'up', 'on', 'the', 'beach', 'not', 'remembering',
'who', 'he', 'was.', '<EOS>', 'John', 'looked', 'at', 'his', 'belongings.',
'<EOS>', 'John', 'found', 'a', 'car', 'key.', '<EOS>', 'John', 'used', 'the',
'car', 'key', 'to', 'find', 'out', 'the', 'car.', '<EOS>', 'John', 'remembered',
'who', 'he', 'was', 'from', 'his', 'things', 'in', 'the', 'car.', '<EOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>',
'<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>', '<SOS>']


total loss:  8.668418261718749
```