D:\myprojects\nlp\rnn\rnn_m21.py

```python
# Many-to-one (2-to-1) RNN 예

#### conda create -n word2vec nltk pandas
# conda activate word2vec
#### conda install pytorch torchvision torchaudio pytorch-cuda=12.1 -c pytorch -c nvidia
#       또는 conda install pytorch torchvision torchaudio cpuonly -c pytorch
# (word2vec) python rnn_m21.py train 500    # 학습시
# (word2vec) python rnn_m21.py i love    # Prediction 시

import numpy as np
import nltk
import torch, sys
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
from tqdm import tqdm

#=======================================================
# Input Target batch 데이터 생성 함수
# #=======================================================
def make_batch(sentences, forTrain=True):
  input_batch = []

  if forTrain :
    target_batch = []

    for sen in sentences:
      words = sen.split()
      input = [word_dict[n] for n in words[:-1]]
      target = word_dict[words[-1]]

          # input번째 자리만 1이 되게 One-Hot Encoding
          # input list 가 2개 값만 가지면 아래는 2개의 one-hot vectors 생성
      input_batch.append(np.eye(n_class)[input])

      target_batch.append(target)

    return input_batch, target_batch

  else : # for Prediction
    sentence = sentences

    words = sentence.split()
    #print(f'words: {words}')
    input = [word_dict[n] for n in words[:-1]]

    input_batch.append(np.eye(n_class)[input])  # One-Hot Encoding

    return input_batch


#=======================================================
# windows data creation
#=======================================================
def make_windows(cleaned_sentences, windowSize=2) :
  MASK_TOKEN = "<MASK>"

```

```
          # 각 문장의 토큰 리스트를 순회하면서 지정된 크기(길이 5개)의 window로 묶어주기
          # lambda 매개변수 : 표현식의 예:
          # matrix = [[1, 2], [3, 4], [5, 6]]
          # squared = [num ** 2 for row in matrix for num in row]
          # print(squared)  # Output: [1, 4, 9, 16, 25, 36]
          # 2차원-->1차원으로 펴기
  flatten = lambda outer_list: [item for inner_list in outer_list for item in inner_list]
          # nltk.ngrams([1,2,3,4,5], 3)) ===> [(1, 2, 3), (2, 3, 4), (3, 4, 5)]
          # 전체 cleaned_sentenses로부터,

          # cleaned_sentences로부터 가져온 각 문장에서 5개의 단어(토큰)로 이루어진 windowx 리스트를 가져와라

          # cleaned_sentences에서 sentence를 하나씩 가져와서 nltk.ngram을 통해
          # sentence 맨 앞과 맨 뒤에 각각 2개씩의 <MASK> 토큰을 추가한 후, 5개 단어 길이의 wind리스트로
          # 바꾼 후, flatten 하라
          # I iove you --> [<MASK>, <MASK>, 'I', 'love', 'you',<MASK>, <MASK>]
  windows = flatten(nltk.ngrams(sentence[:-windowSize].split(' '),windowSize + 1) \
                                    for sentence in tqdm(cleaned_sentences))
  return windows

#=======================================================
# TextRNN Class
#=======================================================
class TextRNN(nn.Module):
  forward_pass = 0

  def __init__(self):
    super(TextRNN, self).__init__()

          # dropout 옵션은 hidden layer갯숫가 1 초과일때만 사용
    #self.rnn = nn.RNN(input_size=n_class, hidden_size=n_hidden, dropout=0.3)

          # input_size=n_class : 입력값을 클래스 크기만큼 길이의 one-hot vector로 할거니까
          # hidden_size : hidden layer node 수 = 5 로 미리 정함
    self.rnn = nn.RNN(input_size=n_class, hidden_size=n_hidden)
          # trainable weight matrix initialized with random values
          # nn.Parameter() : 해당 파라미터가 학습시에 참여하는 weight가 됨
    self.W = nn.Parameter(torch.randn([n_hidden, n_class]).type(dtype)) # n_hiddex * n_class 크기 matrix
          # n_class 크기 bias용 vector
    self.b = nn.Parameter(torch.randn([n_class]).type(dtype))
          # raw 출력값들에 적용할 함수
    self.Softmax = nn.Softmax(dim=1)

  def add_count (self, n=1):
    self.forward_pass += n

  def reset_count (self):
    self.forward_pass = 0

      # X : 입력 시퀀에서의 변하는 X값
      # hidden : 계속 update 되는 hidden matrix 값
      # 이 forward 함수는 epoch 횟수만큼 호출,
  def forward(self, hidden, X):
          # input tensor to have a shape of (seq_len, batch, feature),
          # batch와 seq_length 위치를 교체
          #          이 예제의 경우 (seq_len, batch_size, hidden_size)형식으로 리턴되는데
          #           (2, 11, 5) 크기가 됨
    X = X.transpose(0, 1)
```

```
116           # hidden_vectors: 각 time step 마다의 hidden vector 노드들의 값(h(t))들 저장
117           #           나중에 back-propagation시에 gradient 값 계산을 위하여 저장해 둠
118           #           (seq_len, batch, hidden_size) = (2, 11, 5)
119           # hidden : 마지막 time step 직후의  hidden vector 값
120           #           hidden layer가 다층일 경우에는 layer 층의 갯수 만큼 값을 가짐
121
122
123         # 한번의 호출로 한 batch, 모든 시퀀스, 모든 히든벡터에 대한 벡터값들을 리턴
124         #   2 시퀀스 * 11 batch * 5 vector nodes 값을
125     hidden_vectors, hidden = self.rnn(X, hidden)    # 입력층 batch 값들과 히든 벡터를 이용하여
126                                                     # 히든 벡터 값들을 계산
127     print(f'\n*** hidden_vectors[{self.forward_pass}]: {hidden_vectors}')
128     print(f'\n*** hidden[{self.forward_pass}]: {hidden}')
129
130     last_hidden_vector = hidden_vectors[-1]   # many-to-one 모델이므로 hidden 노드중 마지막 노드값만 필요
131
132             # torch.mm() :  matrix multiplication, 실제 계산 출력값을 여기서 리턴
133     model = torch.mm(last_hidden_vector, self.W) + self.b
134
135     self.add_count(1)
136
137     return model
138
139   def print_output_softmax(self, output):
140         softmax_values = self.Softmax(output)
141         print("\n*** Softmax Results:", softmax_values)
142
143 #=====================================================
144 # Rnn Training 함수
145 #=====================================================
146 def train_rnn(totalEpoch) :
147   criterion = nn.CrossEntropyLoss() # 손실함수를 CrossEntropy로 선택
148   optimizer = optim.Adam(model.parameters(), lr=0.01) # Optimizer를 Adam으로 선택
149
150   #---------------학습 500회
151   for epoch in tqdm(range(totalEpoch)):
152             # requires_grad=True: backpropagation시에 gradients 누적하라
153             # back propagation 시에 batch_size * sequence_length 만큼 gradient를 계산 및 반영하도록
154             #   hidden vector들의 변수가 필요 (0으로 초가화)
155     hidden = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
156
157     output = model(hidden, input_batch)
158
159     #if epoch == totalEpoch-1 :
160     print(f'\n*** Epoch # {epoch}일 때 학습 직후 배치데이터 전체에 대한 output layer 출력값 output:\n{output}')
161     print('-'*80)
162
163     loss = criterion(output, target_batch)
164
165     if totalEpoch <= 3 or (epoch % 100) == 0:
166       print('Epoch:', '%04d' % (epoch), 'cost =', '{:.6f}'.format(loss))
167
168     optimizer.zero_grad()
169     loss.backward()
170     optimizer.step()
171
172 #=====================================================
```

```python
# [1] Word Processing
#=======================================================
sentences = ["i like dogs", "i love coffee", "i hate milk", "i enjoy tea",
  "you like cats", "you love milk", "you hate coffee", "you want snowing",
   "he expects promotion", "he wants money", "he waits snowing"]
windowSize = 2
dtype = torch.float

word_list = list(set(" ".join(sentences).split()))
word_list.sort()    # 매 실행시마다 동일한 word index 번호를 갖게

word_dict = {w: i for i, w in enumerate(word_list)}
print('\n','='*80)
print("word dictionary:",word_dict,'\n')
number_dict = {i: w for i, w in enumerate(word_list)}
n_class = len(word_dict)

#=======================================================
# [2] Input, Target batch data 생성
#=======================================================
      # sentence 문장들로 부터 입력tensor, 출력tensor 구함
input_batch, target_batch = make_batch(sentences, True)
input_batch = torch.tensor(np.array(input_batch), dtype=torch.float32, requires_grad=True)
print('\n','='*80)
print("학습용 input batch:",input_batch)

target_batch = torch.tensor(target_batch, dtype=torch.int64)
print('\n','='*80)
print("학습용 target batch:",target_batch)

#=======================================================
# [3] TextRNN model 생성 및 학습
#=======================================================
            # 저장해야할 hidden matrix의 줄 수는 입력 시퀀스 길이와 같아야 한다
            #   왜냐하면 "입력시퀀스 크기"="bqtch_size" 이고, batch 회수 만큼의 입력 후에 gradient를 업데이트 하니까
batch_size = len(sentences)
n_hidden = 5  # 은닉층의 길이
model = TextRNN()                 # rnn 모델 객체 생성

#=======================================================
# main
#=======================================================
if __name__ == "__main__" :
  print('\n','='*80)
  print('\nUsage [학습]: python rnn_m21.py train 100')
  print('Usage [Prediction]: python rnn_m21.py word1 word2\n')
        # [4] textRNN 모델 학습
  if len(sys.argv) >= 3 :
      #-------------------------------- 학습
    if 'train' in sys.argv[1] and int(sys.argv[2])> 0:
      train_rnn(int(sys.argv[2]))

      # ------------------------------
      # 전체 학습 문장에 대해 prediction 해보기
            # 전체 문장에 대해 입력 벡터 생성
      # ------------------------------
      print('\n','='*80)
      print('학습 완료 했으니, 전체 입력 데이터에 대해, prediction으로 학습 결과 확인')
      model.reset_count()    # hidden_vectors 의 출력 index를 0으로 reset
```

```
      input = [sen.split()[:2] for sen in sentences]

            # hidden layer 값 초기화
            # 1 : hidden layer 수
            # batch_size : 저장할 hidden matrix의 행, 즉, backward propagation 대상이 되는 time 수
            # n_hidden : hidden matrix의 열, 즉 hidden vector 한개의 노드 수
      hidden = torch.zeros(1, batch_size, n_hidden, requires_grad=True)
            # input 값에 대한 prediction 실행
      predict = model(hidden, input_batch).data.max(1, keepdim=True)[1]
            # 결과 출력
      print('\n','.'*80, '\n[학습후 테스트 결과]\n')
      print([sen.split()[:2] for sen in sentences], '->', [number_dict[n.item()] for n in predict.squeeze()])
            # 학습 결과 파일로 저장
      torch.save(model.state_dict(), 'text_rnn.pth')

    else :
      #---------------------------------- 매개변수 문장에 대해 Prediction
      queryStr = sys.argv[1] +' '+ sys.argv[2] + ' ' + sys.argv[2]
      sentence = []
      sentence.append(queryStr)

      #query_input_batch = make_batch(sentence,False)
      query_input_batch = make_batch(queryStr,False)
      query_input_batch = torch.tensor(np.array(query_input_batch), dtype=torch.float32, requires_grad=False)

      print('\n','-'*80)
      print("*** Prediction용 input batch:\n",query_input_batch,'\n')

      model = TextRNN()  # Initialize the model the same way you did before
      model.load_state_dict(torch.load('text_rnn.pth')) # 저장된 파일로 부터 모델 및 파라미터  불러오기
      model.eval()

      hidden = torch.zeros(1, 1, n_hidden, requires_grad=False)
            # input 값에 대한 prediction 후 output layer 출력값(softmax값) 출력해보기
      output = model(hidden, query_input_batch)
      model.print_output_softmax(output)  # 참고로 출력값 찍어보기
            # 답만 가져오기
      #predict = model(hidden, query_input_batch).data.max(1, keepdim=True)[1]
      predict = output.data.max(1, keepdim=True)[1]
            # 결과 출력
      print('\n','-'*80)
      print(f'전체 sentences : {sentences}\n')
      print('\n','-'*80)
      print(f'predict 결과:{predict}')
      print(f'*** {sys.argv[1]}  {sys.argv[2]} ==> {number_dict[predict.item()]}')
```