D:\myprojects\nlp\transformer\self_attention.py

```python
# Vanilla Self Attention

# conda activate word2vec
# python self_attention.py

import torch
import torch.nn.functional as F
import numpy as np
import math

#=====================================================
# 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()
      inputNdx = [wordDictionary[n] for n in words[:-1]]
      target = wordDictionary[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(noOfClasses)[inputNdx])

      target_batch.append(target)

    return input_batch, target_batch

  else : # for Prediction
    sentence = sentences

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

    input_batch.append(np.eye(noOfClasses)[inputNdx])  # One-Hot Encoding

    return input_batch

#=====================================================
# Self-Attention Class
# #=====================================================
class SelfAttention(torch.nn.Module):
    def __init__(self, embeddingSize, headSize):
        super(SelfAttention, self).__init__()
        self.embeddingSize = embeddingSize
        self.headSize = headSize
        self.headDimension = embeddingSize // headSize

        assert (
            self.headDimension * headSize == embeddingSize
        ), "Embedding size needs to be divisible by headSize"

```

```
        #  query, key, value를 scalar값이 아닌 2차원 행렬 형태로 처리
        # 그래서 이 샘플 코드에서는 2차원 weighr표현에 좋은 nn.Linear 형식을 사용
        # 만약, outer product 가 아닌 dot product를 할 경우엔 vector가 필요함.
        self.values = torch.nn.Linear(self.headDimension, self.headDimension, bias=False)
        print(f'\nvalues 벡터 입출력 크기: {self.values}')
        self.keys = torch.nn.Linear(self.headDimension, self.headDimension, bias=False)
        self.queries = torch.nn.Linear(self.headDimension, self.headDimension, bias=False)
        self.fc_out = torch.nn.Linear(headSize * self.headDimension, embeddingSize)

    def forward(self, valuesBatch, keysBatch, queriesBatch):
        batchSize = queriesBatch.shape[0]   # sentence 갯수와 동일 => 12
        valueLength, keyLength, queryLength = valuesBatch.shape[1], keysBatch.shape[1], queriesBatch.shape[1]
        print(f'\nbatchSize: {batchSize}\n [query, key, value] 의 시퀀스 길이: \
                [{valueLength}, {keyLength}, {queryLength}] \n')

        # Split the embedding into self.headSize different pieces
        queriesBatchMultiHead = queriesBatch.reshape(batchSize, queryLength, self.headSize, self.headDimension)
        keysBatchMultiHead = keysBatch.reshape(batchSize, keyLength, self.headSize, self.headDimension)
        valuesBatchMultiHead = valuesBatch.reshape(batchSize, valueLength, self.headSize, self.headDimension)

        queriesBatch = self.queries(queriesBatchMultiHead)
        keysBatch = self.keys(keysBatchMultiHead)
        valuesBatch = self.values(valuesBatchMultiHead)

        # q * t 를 하여 similarity(attention 값)을 구함,
        # simililarity 계산법은 정하기 나름인데, 여기서는 outer product(행렬곱)를 수행하므로
        # attention 결과는 4차원 tensor [batch, head, queryDimension, keyDimension]
        attention = torch.einsum("nqhd,nkhd->nhqk", [queriesBatch, keysBatch])
        # queries shape: (N, queryLength, headSize, head_dim),
        # keysBatch shape: (N, keyLength, headSize, head_dim),
        # attention: (N, headSize, queryLength, keyLength)

        # Scale attention scores
        attention = attention / (self.embeddingSize ** (1/2))
        attention = F.softmax(attention, dim=-1)

        # 방금 계산한 attention과 values를 곱하여 출력값 계산,
        # 그 후 embeddingSize 사이즈를 갖도록 한 차원 축소된 shape 변경
        out = torch.einsum("nhql,nlhd->nqhd", [attention, valuesBatch]).reshape(
            batchSize, queryLength, self.headSize * self.headDimension
        )
        # attention shape: (N, headSize, queryLength, keyLength)
        # valuesBatch shape: (N, valueLength, headSize, head_dim)
        # after einsum: (N, queryLength, headSize, head_dim) then we flatten the last two dimensions

        out = self.fc_out(out)  # (N, queryLength, embeddingSize) 크기로 리턴
        print(f'attention 후 출력 y: {out}')
        return out

#=====================================================================
sentences = ["i like my dogs", "i love hot coffee", "i hate cold milk", "i enjoy hot tea",
  "you like cute cats", "you love hot milk", "you hate cold coffee", "you want rainy days",
   "he expects big promotion", "he wants nice gifts", "he wants my dogs", "i expect rainy days"]

dtype = torch.float
```

```
113
114 wordList = list(set(" ".join(sentences).split()))
115 wordList.sort()    # 매 실행시마다 동일한 word index 번호를 갖게
116
117 wordDictionary = {w: i for i, w in enumerate(wordList)}
118 print('\n','='*80)
119 print("word dictionary:",wordDictionary,'\n')
120 number_dict = {i: w for i, w in enumerate(wordList)}
121
122 windowSize = 3
123 headSize = 4
124 noOfClasses = int(math.ceil(len(wordDictionary) / headSize) * headSize)
125 embeddingSize = noOfClasses
126
127 print(f'\nheadSize: {headSize}, noOfClasses: {noOfClasses}')
128
129 # make one-hot encoded input batch
130 input_batch, target_batch = make_batch(sentences,True)
131 valueLength, keyLength, queryLength = windowSize, windowSize, windowSize # sequence
    lengths => 3
132
133 queries = torch.tensor(np.array(input_batch), dtype=torch.float32)
134 keys = torch.tensor(np.array(input_batch), dtype=torch.float32)
135 values = torch.tensor(np.array(input_batch), dtype=torch.float32)
136 print('\n','='*80)
137 print(f"Self-Attention용 [{len(queries)}]개 input batch:",input_batch)
138
139 attention = SelfAttention(embeddingSize, headSize)
140
141 # Self-attention(Forward Pass only) 실행
142 out = attention(values, keys, queries)
143
144 print(f'\noutput의 shape[batchSize, sequenceLength, VectorDimension]: {out.shape}\n') #
    Expected shape: (batchSize, sequenceLength,embeddingSize)
145 print('첫 번째 batch의 앞 쪽 2개의 시퀀스에 대한 attention 결과값 출력해보기')
146 for i in range(2):
147     print('-'*80)
148     for j in range(out.shape[1]):
149         print(f'sequence #[{i}][{j}] {sentences[i].split()[j]}\n')
150         for k in range(out.shape[2]):
151             print(out[i, j, k].item(), end=' ')
152         print('\n')
153     print('\n')
154
155
```