RNN (Recurrent Neural Network)

(https://towardsdatascience.com/rnns-from-theory-to-pytorch-f0af30b610e1)

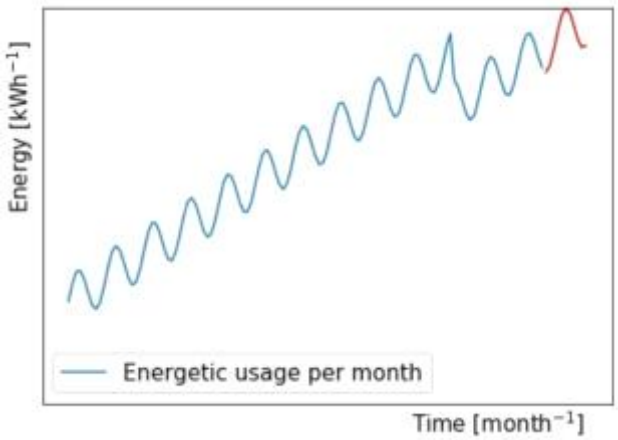
(http://cs231n.stanford.edu/slides/2023/lecture_8.pdf)

◉ RNN : Sequence 데이터의 처리에 적합한 신경망 모델의 원조

◉ A Sequence : a series of related things or events
a set of data that contains trainable context

◉ RNN으로 해결 가능한 문제의 예 2개

(1) Timeseries Forecasting



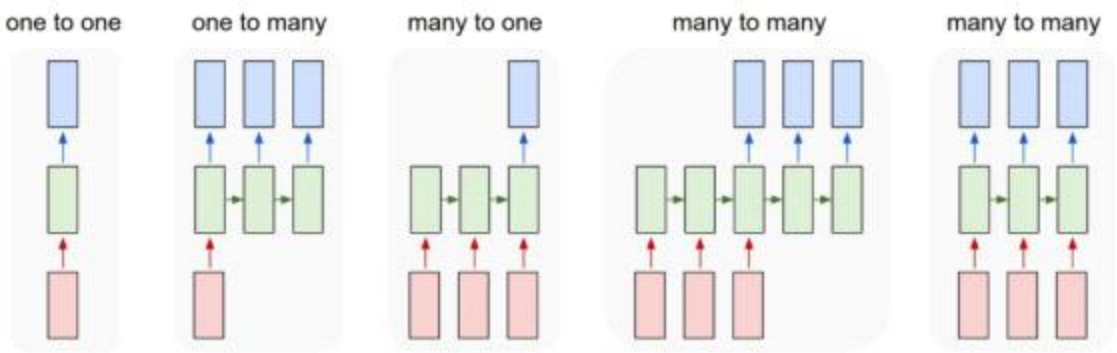
(2) Natural Language Processing

Mary rides the bicycle, the bycicle is ____.

hers

(instead of his, theirs)

◉ 입력, 출력에 따른 RNN의 종류와 사용 예



용례		Image Captioning	Action Prediction	Video Captioning	Video(Frame) Classificaiton
입력		Image	Video frames	Video frames	Video frames
출력		Words seq.	Action class	Caption	Frame classification

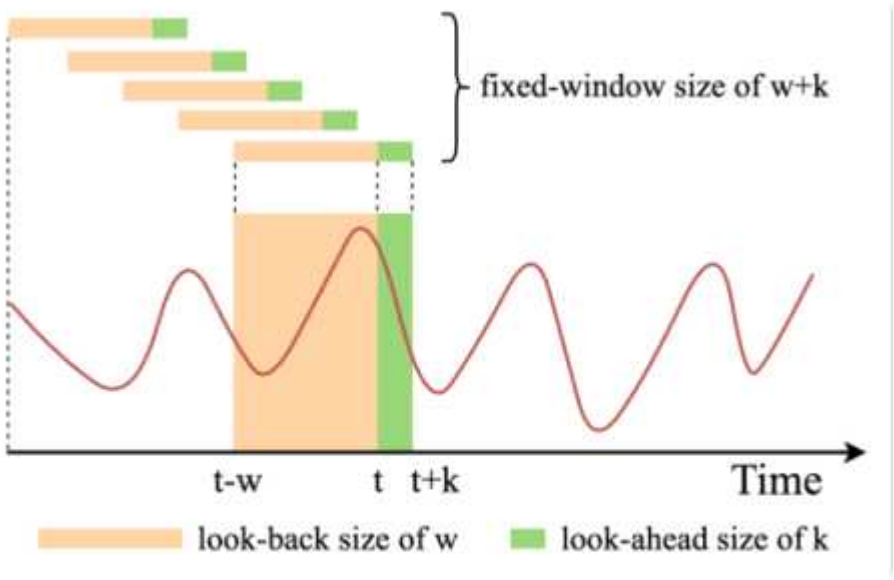
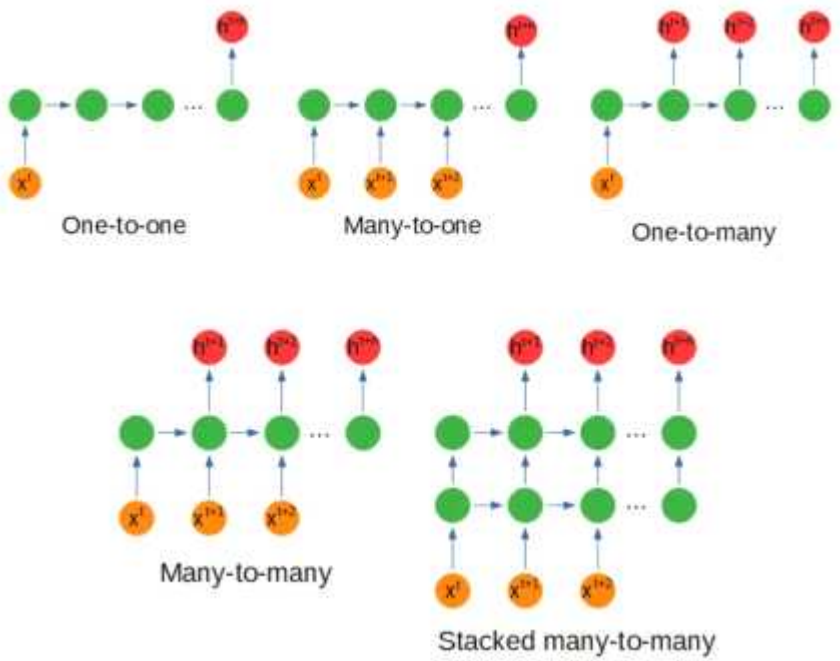


Fig. 1 Using a sliding window of width w to construct the training set for predicting k time steps in the future.

(https://arxiv.org/ftp/arxiv/papers/2204/2204.11115.pdf)



(1) Many-to-one :

예) 영화장면의 Sequence -> 1개의 감정표현

여러 단어의 싯구 -> 1개의 감정

예) 입력 숫자 3개, 출력 숫자 1개

[학습 데이터] 10 20 30 ------> 40

20 30 40 ------> 50

.....

60 70 80 ------> 90

[테스트 데이터]

100, 110, 120 ------> ??

(2) One-to-Many :

예) 1개의 감정 단어 -> 여러 단어로 이루어진 시 문장

(3) Many-to-Many :

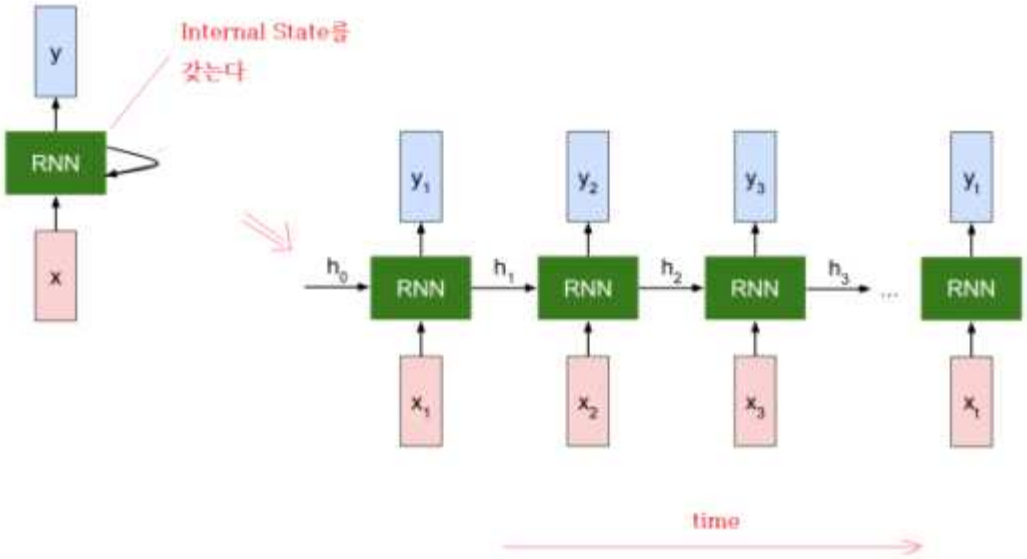
예) 12개월의 과거 전력 사용량 값들 -> 다음 6개월의 전력 사용량 값들

영어 문장 단어들 -> 한국어 문장 단어들

(4) Stacked Many-to-many :

Hidden states가 여러 층으로 구성된 Many-to_many RNN

◉ RNN의 Hidden State, Output 계산



$h_t = f_W(h_{t-1}, x_t)$

new state　old state　input vector at some time step

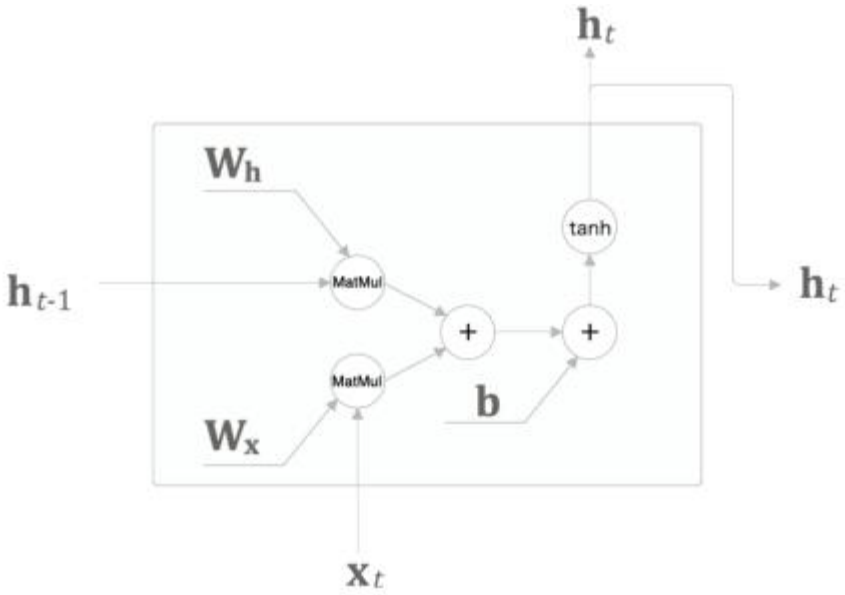
some function with parameters W

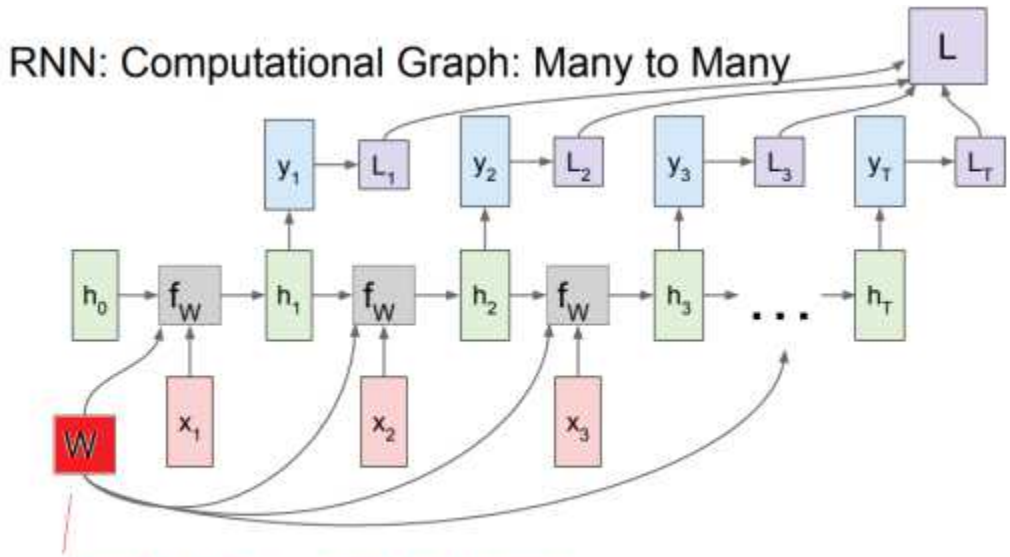
$y_t = f_{W_{hy}}(h_t)$

output　new state

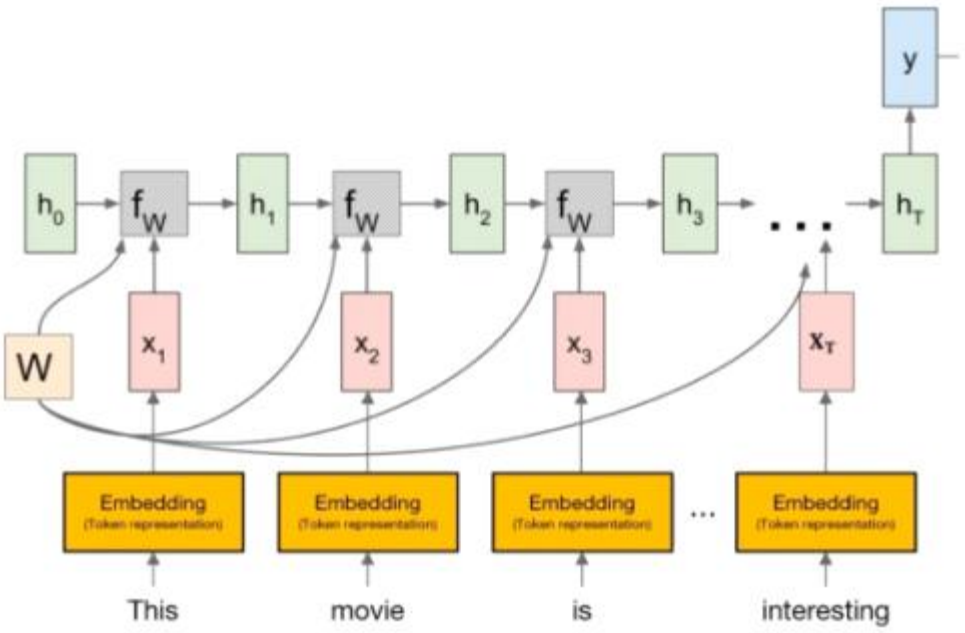
another function with parameters W_o

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$





하나의 Weight가 time에 따라 계속 사용된다



◎ Pytorch RNN Parameter

(https://pytorch.org/docs/stable/generated/torch.nn.RNN.html)

RNN

CLASS torch.nn.RNN(*self, input_size, hidden_size, num_layers=1, nonlinearity='tanh', bias=True, batch_first=False, dropout=0.0, bidirectional=False, device=None, dtype=None*) [SOURCE]

Apply a multi-layer Elman RNN with $\tanh$ or ReLU non-linearity to an input sequence. For each element in the input sequence, each layer computes the following function:

$$h_t = \tanh(x_t W_{ih}^T + b_{ih} + h_{t-1} W_{hh}^T + b_{hh})$$

where h_t is the hidden state at time *t*, x_t is the input at time *t*, and $h_{(t-1)}$ is the hidden state of the previous layer at time *t-1* or the initial hidden state at time *0*. If `nonlinearity` is `'relu'`, then ReLU is used instead of $\tanh$.

Parameters

- **input_size** – The number of expected features in the input *x*
- **hidden_size** – The number of features in the hidden state *h*
- **num_layers** – Number of recurrent layers. E.g., setting `num_layers=2` would mean stacking two RNNs together to form a *stacked RNN*, with the second RNN taking in outputs of the first RNN and computing the final results. Default: 1
- **nonlinearity** – The non-linearity to use. Can be either `'tanh'` or `'relu'`. Default: `'tanh'`
- **bias** – If `False`, then the layer does not use bias weights *b_ih* and *b_hh*. Default: `True`
- **batch_first** – If `True`, then the input and output tensors are provided as (*batch, seq, feature*) instead of (*seq, batch, feature*). Note that this does not apply to hidden or cell states. See the Inputs/Outputs sections below for details. Default: `False`
- **dropout** – If non-zero, introduces a *Dropout* layer on the outputs of each RNN layer except the last layer, with dropout probability equal to `dropout`. Default: 0
- **bidirectional** – If `True`, becomes a bidirectional RNN. Default: `False`

Inputs: input, h_0

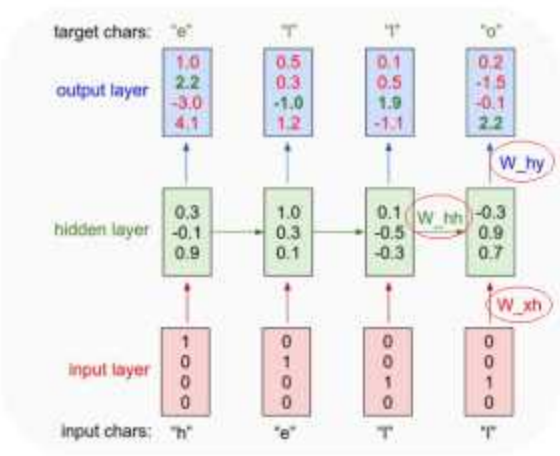
- **input**: tensor of shape (L, H_{in}) for unbatched input, (L, N, H_{in}) when `batch_first=False` or (N, L, H_{in}) when `batch_first=True` containing the features of the input sequence. The input can also be a packed variable length sequence. See `torch.nn.utils.rnn.pack_padded_sequence()` or `torch.nn.utils.rnn.pack_sequence()` for details.
- **h_0**: tensor of shape $(D * \text{num\_layers}, H_{out})$ for unbatched input or $(D * \text{num\_layers}, N, H_{out})$ containing the initial hidden state for the input sequence batch. Defaults to zeros if not provided.

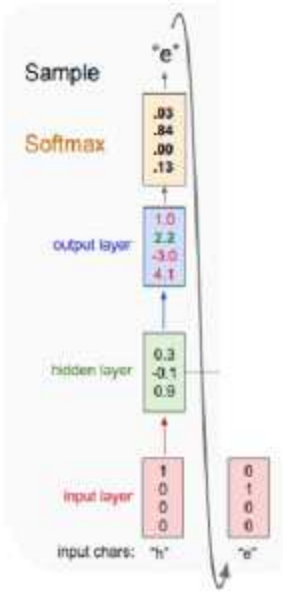
where:

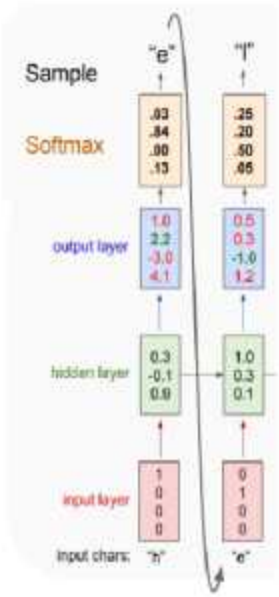
$$\begin{aligned} N &= \text{batch size} \\ L &= \text{sequence length} \\ D &= 2 \text{ if bidirectional=True otherwise } 1 \\ H_{in} &= \text{input\_size} \\ H_{out} &= \text{hidden\_size} \end{aligned}$$

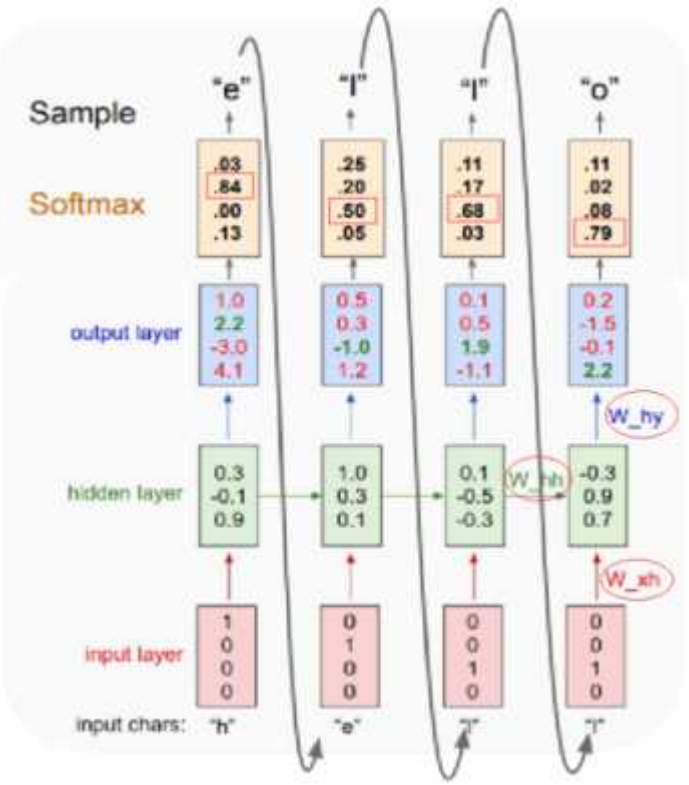
◎ Character-Level의 Language Model 의 예 (1)

- Vocabulary : [h,e,l,o] 이고,
- 실행(test time)시에 한 character씩 다음 character를 출력하는 예





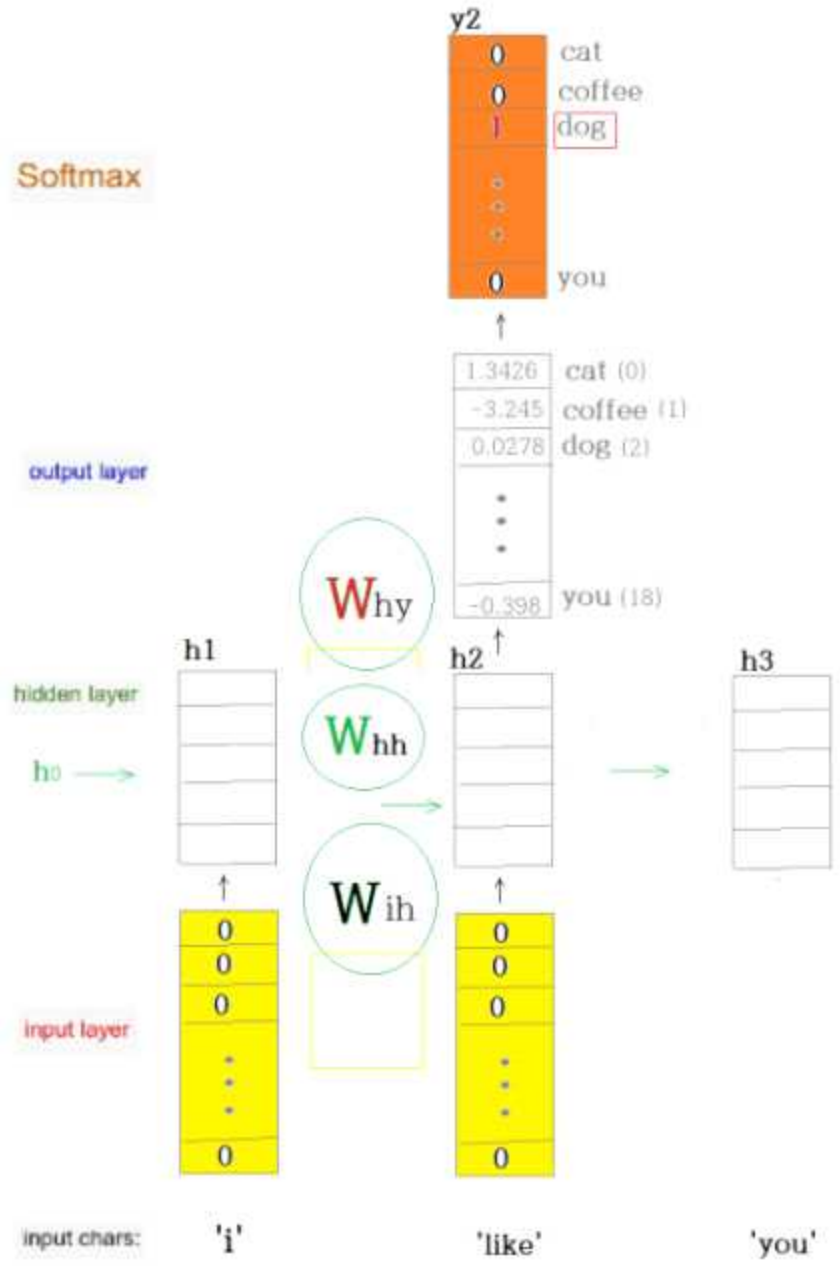




◎ Word-Level의 Language Model 의 예 (2)

- Vocabulary : {'cat': 0, 'coffee': 1, 'dog': 2, 'enjoy': 3, 'expects': 4, 'hate': 5, 'he': 6, 'i': 7, 'like': 8, 'love': 9, 'milk': 10, 'money': 11, 'promotion': 12, 'snowing': 13, 'tea': 14, 'waits': 15, 'want': 16, 'wants': 17, 'you': 18}

- 2개의 단어 다음에 나올 단어 한 개를 예측 (2-to-1 model)



* 실제 학습시에는 Hidden Vector는 (배치크기 * 시퀀스 길이 * 벡터 크기) 만큼의 공간이 필요
3개의 W matrix는 t에 상관없이 overwrite(update) 되므로, W는 3개만 필요함
반면, hidden vector는 (batch_size * sequence_length) 개 만큼 필요함

◉ RNN 계열 모델들의 장점 :

- 시퀀스 앞 부분의 정보를 t 시점에서 활용한다 (그래서 시퀀스 처리 모델임)

- 입력 시퀀스 길이에 거의 관계 없이 모델 사이즈가 일정하다

- 이론적으로는 긴 입력 시퀀스도 처리 가능하다

◉ RNN 계열 모델들의 문제점 :

- 기울기 소실 (Gradient Vanishing), Gradient Exploding 둘 다 문제임

- 순차적 처리로 병렬처리가 어려움, 학습시간이 김

- 장기 의존성 문제 (Long-term Dependency):
  실제적으로는 현재 시점에서 멀리 떨어진 시점의 정보를 access 하는 것이 힘듦

- LSTM(Long Short-term Memory), GRU(Gated Recurrent Unit) 같은 개선 버전이 있으나
  모델이 복잡하고, 큰 데이터셋이나 복잡한 시퀀스에 취약

=> 시퀀스가 아주 짧고 단순한 경우를 제외하고 Transformer모델로 대체

[1] Two-to-one Language Model RNN Example

(https://justkode.kr/deep-learning/pytorch-rnn/)

```
# 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
importnumpy asnp
importnltk
importtorch,sys
importtorch.nn asnn
importtorch.optim asoptim
importtorch.nn.functional asF
fromtqdm importtqdm
#=====================================================
# Input Target batch 데이터 생성 함수
# #=====================================================
def make_batch(sentences,forTrain=True):
input_batch =[]

  ifforTrain :
target_batch =[]
    forsen insentences:
words =sen.split()
      input =[word_dict[n]forinwords[:-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)

    returninput_batch,target_batch

  else :# for Prediction
sentence =sentences

words =sentence.split()
    #print(f'words: {words}')
    input =[word_dict[n]forinwords[:-1]]
input_batch.append(np.eye(n_class)[input])  # One-Hot Encoding

    returninput_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 forinner_list inouter_list foritem ininner_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 love you --> [<MASK>, <MASK>, 'I', 'love', 'you',<MASK>, <MASK>]
windows =flatten(nltk.ngrams(sentence[:-windowSize].split(''),windowSize +1)\
                                  forsentence intqdm(cleaned_sentences))
  returnwindows
```

```
#=====================================================
# 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)
          # hidden_vectors: 각 time step 마다의 hidden vector 노드들의 값(h(t))들 저장
          #           나중에 back-propagation시에 gradient 값 계산을 위하여 저장해 둠
          #           (seq_len, batch, hidden_size) = (2, 11, 5)
          # hidden : 마지막 time step 직후의  hidden vector 값
          #           hidden layer가 다층일 경우에는 layer 층의 갯수 만큼 값을 가짐


        # 한번의 호출로 한 batch, 모든 시퀀스, 모든 히든벡터에 대한 벡터값들을 리턴
        #   2 시퀀스 * 11 batch * 5 vector nodes 값을
hidden_vectors,hidden =self.rnn(X,hidden)    # 입력층 batch 값들과 히든 벡터를 이용하여
                                             # 히든 벡터 값들을 계산
    print(f'\n*** hidden_vectors[{self.forward_pass}]: {hidden_vectors}')

    print(f'\n*** hidden[{self.forward_pass}]: {hidden}')
last_hidden_vector =hidden_vectors[-1]   # many-to-one 모델이므로 hidden 노드중 마지막 노드값만 필요
           # torch.mm() :  matrix multiplication, 실제 계산 출력값을 여기서 리턴
model =torch.mm(last_hidden_vector,self.W)+self.b

    self.add_count(1)

    returnmodel
  def print_output_softmax(self,output):
softmax_values =self.Softmax(output)
        print("\n*** Softmax Results:",softmax_values)

#=====================================================
# Rnn Training 함수
#=====================================================
def train_rnn(totalEpoch):
criterion =nn.CrossEntropyLoss()# 손실함수를 CrossEntropy로 선택
optimizer =optim.Adam(model.parameters(),lr=0.01)# Optimizer를 Adam으로 선택
  #---------------학습 500회
  forepoch intqdm(range(totalEpoch)):
           # requires_grad=True: backpropagation시에 gradients 누적하라
           # back propagation 시에 batch_size * sequence_length 만큼 gradient를 계산 및 반영하도록
           #   hidden vector들의 변수가 필요 (0으로 초기화)
hidden =torch.zeros(1,batch_size,n_hidden,requires_grad=True)

output =model(hidden,input_batch)
```

```
    #if epoch == totalEpoch-1 :
       print(f'\n***  Epoch  #  {epoch}일  때  학습  직후  배치데이터  전체에  대한  output  layer  출력값
output:\n{output}')
    print('-'*80)
loss =criterion(output,target_batch)
    iftotalEpoch <=3 or (epoch %100)==0:
      print('Epoch:','%04d'%(epoch),'cost =','{:.6f}'.format(loss))

optimizer.zero_grad()
loss.backward()
optimizer.step()
#=====================================================
# [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:fori,in enumerate(word_list)}
print('\n','='*80)
print("word dictionary:",word_dict,'\n')
number_dict ={i:fori,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"="batch_size" 이고, batch 회수 만큼의 입력 후에 gradient를 업데이트 하
니까
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'insys.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]forsen insentences]
          # 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]forsen insentences],'->',[number_dict[n.item()]forinpredict.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()]}')
```

[2] 실행 결과 화면 (epoch = 2인 경우의 학습 시)

```
(word2vec) D:\myprojects\nlp\rnn>python rnn_m21.py train 2

 ================================================================================
word dictionary: {'cats': 0, 'coffee': 1, 'dogs': 2, 'enjoy': 3, 'expects': 4, 'hate': 5, 'he': 6, 'i': 7, 'like': 8, 'love': 9,
'milk': 10, 'money': 11, 'promotion': 12, 'snowing': 13, 'tea': 14, 'waits': 15, 'want': 16, 'wants': 17, 'you': 18}


 ================================================================================
input batch: tensor([[[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,          <---- 'i'
          0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,                      <----- 'like'
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,                      <---- 'i'
          0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,                      <----- 'love'
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.],
         [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.],
         [0., 0., 0., 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., 1.],
         [0., 0., 0., 0., 0., 0., 0., 0., 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., 1.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 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., 1.],
         [0., 0., 0., 0., 0., 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., 1.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.],
         [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 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.,
          1., 0.]],

        [[0., 0., 0., 0., 0., 0., 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., 1., 0.,
          0., 0.]]], requires_grad=True)

 ================================================================================
target batch: tensor([ 2,  1, 10, 14,  0, 10,  1, 13, 12, 11, 13])     <----- batch 단위로 update하므로 크기는 배치 크기

          배치 첫 번째 문장의 target인 'dog'의 사전 인덱스        'snowing'의 사전 인덱스

 ================================================================================

Usage [학습]: python rnn_m21.py train 100
Usage [Prediction]: python rnn_m21.py word1 word2
```

```
  0%|
 | 0/2 [00:00<?, ?it/s]
```

Hidden Vector는 (배치크기 * 시퀀스 길이 * 벡터 크기) 만큼의 공간이 필요
따라서, 이 예제에서는 12 * 2 * 5 만큼의 콩간이 필요

<< 첫 번째 EPOCH >>

```
*** hidden_vectors[0]: tensor([[[-0.3216, -0.5452, -0.4059, -0.0968,  0.1629],
         [-0.3216, -0.5452, -0.4059, -0.0968,  0.1629],
         [-0.3216, -0.5452, -0.4059, -0.0968,  0.1629],
         [-0.0372, -0.4483, -0.1498,  0.0155, -0.0091],
         [-0.0372, -0.4483, -0.1498,  0.0155, -0.0091],
         [-0.0372, -0.4483, -0.1498,  0.0155, -0.0091],
         [-0.0372, -0.4483, -0.1498,  0.0155, -0.0091],
         [ 0.3085, -0.0316, -0.4030,  0.4874, -0.2441],
         [ 0.3085, -0.0316, -0.4030,  0.4874, -0.2441],
         [ 0.3085, -0.0316, -0.4030,  0.4874, -0.2441]],

        [[-0.6419, -0.4523,  0.1143,  0.0491,  0.6902],
         [-0.8113, -0.4327, -0.1408,  0.4089,  0.4649],
         [-0.8378, -0.4602, -0.0627, -0.1802,  0.4266],
         [-0.9027, -0.2357, -0.1514, -0.1480,  0.2277],
         [-0.5705, -0.3553,  0.2137,  0.2038,  0.5425],
         [-0.7689, -0.3339, -0.0395,  0.5312,  0.2570],
         [-0.8007, -0.3640,  0.0394, -0.0246,  0.2118],
         [-0.8041, -0.0225,  0.4262,  0.5456,  0.3920],
         [-0.8551,  0.2187,  0.6095,  0.6462,  0.0743],
         [-0.5781,  0.2309,  0.2089,  0.4037, -0.3434],
         [-0.5928,  0.2578,  0.5898,  0.1458,  0.4193]]],
       grad_fn=<StackBackward0>)
```

<------ 첫 번째 시퀀스 word에 대한
11개 batch 벡터값
(시퀀스 첫 번째 단어:'i','you','he')

<------ 두 번째 시퀀스 word에 대한
11 개 batch 벡터값
(시퀀스 두 번째 단어:'love','like','expects' ...)

```
*** hidden[0]: tensor([[[ 0.2026, -0.3466, -0.1333,  0.4393,  0.2556],
         [[-0.6419, -0.4523,  0.1143,  0.0491,  0.6902],
         [-0.8113, -0.4327, -0.1408,  0.4089,  0.4649],
         [-0.8378, -0.4602, -0.0627, -0.1802,  0.4266],
         [-0.9027, -0.2357, -0.1514, -0.1480,  0.2277],
         [-0.5705, -0.3553,  0.2137,  0.2038,  0.5425],
         [-0.7689, -0.3339, -0.0395,  0.5312,  0.2570],
         [-0.8007, -0.3640,  0.0394, -0.0246,  0.2118],
         [-0.8041, -0.0225,  0.4262,  0.5456,  0.3920],
         [-0.8551,  0.2187,  0.6095,  0.6462,  0.0743],
         [-0.5781,  0.2309,  0.2089,  0.4037, -0.3434],
         [-0.5928,  0.2578,  0.5898,  0.1458,  0.4193]]],
       grad_fn=<StackBackward0>)

*** Epoch # 0일 때 학습 직후 배치데이터 전체에 대한 output layer 출력값 output:
tensor([[-6.4833e-01, -2.9322e+00, -1.0049e+00,  4.8337e-01, -6.1329e-01,
          8.1251e-01,  8.4654e-01,  6.8093e-01,  1.3145e+00, -8.1489e-01,
          6.0670e-01, -8.3706e-01, -1.4239e+00,  1.4976e+00,  5.8864e-01,
         -3.7782e-01,  1.2989e+00,  1.5892e+00,  2.3070e+00],
        [-7.7896e-01, -1.4255e+00, -7.4623e-01,  1.4793e-01, -1.3802e+00,
         -1.5912e-01, -3.1743e-02,  1.1281e+00, -5.2364e-02, -8.2708e-01,
         -3.3585e-02, -1.1563e+00, -1.6578e-01,  1.6628e+00,  1.8821e+00,
         -9.8718e-01,  2.0494e+00,  1.8044e-01,  1.8818e+00],
        [-1.3983e-01, -2.1078e+00,  2.5746e-02,  3.2418e-01, -1.3213e+00,
          5.9797e-01,  6.6105e-02,  5.7694e-02,  7.6094e-01, -3.3888e-01,
         -1.6959e-01, -3.3911e-01, -1.4665e+00,  6.4343e-01,  6.1448e-01,
         -1.1721e-01,  1.2287e+00,  2.9005e-01,  2.4833e+00],
        [ 8.4713e-01, -3.0561e+00,  3.3510e-01,  6.3071e-01, -1.9067e+00,
          1.3770e+00,  1.9177e-01,  1.2768e-02,  6.8869e-01, -2.3248e-01,
          6.2754e-01, -3.1910e-01, -7.7025e-01,  1.0212e+00,  1.3338e+00,
         -1.7861e+00,  1.0615e+00,  2.0591e+00,  1.6063e+00],
        [-1.0192e+00, -2.9346e+00, -1.3047e+00,  3.7716e-01, -3.5192e-01,
          6.6778e-01,  9.3493e-01,  9.6906e-01,  1.1621e+00, -8.9816e-01,
          5.6283e-01, -9.6474e-01, -1.3552e+00,  1.6901e+00,  6.9903e-01,
         -1.8911e-01,  1.3973e+00,  1.5615e+00,  2.3382e+00],
        [-1.1473e+00, -1.4429e+00, -1.0580e+00,  5.0107e-02, -1.1259e+00,
         -2.9190e-01,  7.1117e-02,  1.4192e+00, -1.9714e-01, -9.1899e-01,
         -6.5957e-02, -1.2854e+00, -9.6860e-02,  1.8736e+00,  1.9868e+00,
         -8.1909e-01,  2.1460e+00,  1.8468e-01,  1.9104e+00],
        [-5.3831e-01, -2.1364e+00, -3.1332e-01,  2.1790e-01, -1.0424e+00,
          4.5888e-01,  1.8025e-01,  3.7554e-01,  6.0257e-01, -4.3698e-01,
```

<- 출력 벡터 (19개 값)

```
          -2.0079e-01, -4.7861e-01, -1.3902e+00,  8.7401e-01,  7.3158e-01,
           6.2158e-02,  1.3312e+00,  3.0612e-01,  2.5109e+00],
         [ 4.8552e-01, -3.1463e+00, -3.2052e-01,  7.1677e-01, -1.1276e+00,
           1.1193e+00,  5.1561e-01,  4.0880e-01,  1.4897e+00, -6.3281e-01,
           1.3142e+00, -9.8599e-01, -6.9701e-01,  1.3200e+00,  1.3657e+00,
          -1.6998e+00,  1.3174e+00,  2.2577e+00,  1.4332e+00],
         [-1.3133e+00, -4.0400e+00, -1.4056e+00, -2.2649e-03, -6.4476e-01,
           1.2944e+00,  1.0697e+00,  1.6412e+00, -8.1092e-01, -4.6251e-01,
           7.1146e-02, -4.0148e-01, -6.8558e-01,  2.3919e+00,  1.7335e+00,
          -7.6502e-01,  1.1387e+00,  3.0120e+00,  1.9884e+00],
         [-1.0214e+00, -3.0319e+00, -9.1045e-01,  1.8372e-01, -1.9075e-01,
           7.5400e-01,  7.3972e-01,  7.1543e-01,  9.2161e-01, -5.1882e-01,
           1.5404e-01, -5.7331e-01, -1.7213e+00,  1.1494e+00,  5.1697e-01,
           5.0940e-01,  1.1277e+00,  1.1662e+00,  2.5953e+00],
         [-4.6642e-01, -4.1781e+00, -1.2497e+00,  5.5634e-01, -4.7946e-01,
           1.5034e+00,  1.2723e+00,  1.0460e+00,  1.0640e+00, -7.4989e-01,
           1.1997e+00, -8.3166e-01, -1.0273e+00,  2.0602e+00,  1.1370e+00,
          -1.2105e+00,  1.0814e+00,  3.3736e+00,  1.7545e+00]],
        grad_fn=<AddBackward0>)
--------------------------------------------------------------------------------
Epoch: 0000 cost = 3.998404
```

<< 두 번째 EPOCH >>

```
*** hidden_vectors[1]: tensor([[[-0.3483, -0.5238, -0.3806, -0.1264,  0.1920],
         [-0.3483, -0.5238, -0.3806, -0.1264,  0.1920],
         [-0.3483, -0.5238, -0.3806, -0.1264,  0.1920],
         [-0.0671, -0.4240, -0.1203, -0.0144,  0.0209],
         [-0.0671, -0.4240, -0.1203, -0.0144,  0.0209],
         [-0.0671, -0.4240, -0.1203, -0.0144,  0.0209],
         [-0.0671, -0.4240, -0.1203, -0.0144,  0.0209],
         [ 0.2812, -0.0016, -0.3775,  0.4642, -0.2157],
         [ 0.2812, -0.0016, -0.3775,  0.4642, -0.2157],
         [ 0.2812, -0.0016, -0.3775,  0.4642, -0.2157]],

        [[ 0.1371, -0.3167, -0.1297,  0.3769,  0.2646],
         [-0.4078, -0.2612, -0.2096,  0.2396, -0.2395],
         [-0.0488,  0.1138, -0.0618,  0.0886,  0.1769],
         [-0.3986, -0.1013,  0.5273,  0.0930,  0.3090],
         [ 0.2275, -0.3910, -0.2139,  0.4436,  0.1463],
         [-0.3270, -0.3385, -0.2910,  0.3136, -0.3522],
         [ 0.0447,  0.0293, -0.1476,  0.1675,  0.0550],
         [-0.2693, -0.4654,  0.2769,  0.1430,  0.3532],
         [ 0.2506, -0.3860,  0.3115,  0.6992, -0.3463],
         [ 0.4762, -0.1537, -0.1557,  0.3345,  0.0351],
         [ 0.1609, -0.5889,  0.2526,  0.5436,  0.2141]]],
       grad_fn=<StackBackward0>)

*** hidden[1]: tensor([[[ 0.1371, -0.3167, -0.1297,  0.3769,  0.2646],
         [-0.4078, -0.2612, -0.2096,  0.2396, -0.2395],
         [-0.0488,  0.1138, -0.0618,  0.0886,  0.1769],
         [-0.3986, -0.1013,  0.5273,  0.0930,  0.3090],
         [ 0.2275, -0.3910, -0.2139,  0.4436,  0.1463],
         [-0.3270, -0.3385, -0.2910,  0.3136, -0.3522],
         [ 0.0447,  0.0293, -0.1476,  0.1675,  0.0550],
         [-0.2693, -0.4654,  0.2769,  0.1430,  0.3532],
         [ 0.2506, -0.3860,  0.3115,  0.6992, -0.3463],
         [ 0.4762, -0.1537, -0.1557,  0.3345,  0.0351],
         [ 0.1609, -0.5889,  0.2526,  0.5436,  0.2141]]],
       grad_fn=<StackBackward0>)

*** Epoch # 1일 때 학습 직후 배치데이터 전체에 대한 output layer 출력값 output:
tensor([[-4.8692e-01, -2.7425e+00, -8.2792e-01,  4.7641e-01, -7.3864e-01,
          7.2247e-01,  6.9434e-01,  5.6740e-01,  1.3308e+00, -8.0572e-01,
          6.1457e-01, -8.3047e-01, -1.3536e+00,  1.3891e+00,  6.3840e-01,
         -4.5871e-01,  1.3231e+00,  1.3668e+00,  2.2397e+00],
        [-6.4890e-01, -1.2065e+00, -6.0693e-01,  1.5485e-01, -1.4448e+00,
         -2.6766e-01, -1.6749e-01,  1.0322e+00,  2.8342e-02, -8.2432e-01,
          1.6299e-02, -1.2038e+00, -9.5441e-02,  1.5498e+00,  1.9359e+00,
         -1.0342e+00,  2.0847e+00, -5.7576e-02,  1.8190e+00],
        [ 3.1174e-03, -1.8821e+00,  1.7645e-01,  3.2934e-01, -1.4002e+00,
          4.7546e-01, -8.5855e-02, -3.5786e-02,  8.3950e-01, -3.4156e-01,
```

```
          -1.3273e-01, -3.9180e-01, -1.3885e+00,  5.2216e-01,  6.8483e-01,
          -1.7912e-01,  1.2868e+00,  2.9669e-02,  2.4120e+00],
         [ 9.7434e-01, -2.8744e+00,  4.8450e-01,  6.1153e-01, -1.9696e+00,
           1.2678e+00,  3.2782e-02, -5.2103e-02,  7.0074e-01, -2.1742e-01,
           6.4882e-01, -3.5911e-01, -6.7168e-01,  9.1882e-01,  1.4502e+00,
          -1.8408e+00,  1.1084e+00,  1.8318e+00,  1.5279e+00],
         [-8.7091e-01, -2.7557e+00, -1.1379e+00,  3.6654e-01, -4.8349e-01,
           5.8595e-01,  7.9294e-01,  8.6044e-01,  1.1554e+00, -8.8953e-01,
           5.5985e-01, -9.4560e-01, -1.2859e+00,  1.5963e+00,  7.4091e-01,
          -2.6983e-01,  1.4143e+00,  1.3566e+00,  2.2774e+00],
         [-1.0256e+00, -1.2243e+00, -9.2459e-01,  5.4029e-02, -1.2090e+00,
          -3.9587e-01, -6.0002e-02,  1.3273e+00, -1.4613e-01, -9.1766e-01,
          -3.0072e-02, -1.3207e+00, -1.9728e-02,  1.7772e+00,  2.0386e+00,
          -8.7671e-01,  2.1780e+00, -4.2025e-02,  1.8511e+00],
         [-4.0492e-01, -1.9151e+00, -1.6674e-01,  2.1878e-01, -1.1343e+00,
           3.4294e-01,  3.3727e-02,  2.8298e-01,  6.5408e-01, -4.3837e-01,
          -1.7953e-01, -5.1557e-01, -1.3124e+00,  7.6379e-01,  7.9383e-01,
          -3.9193e-04,  1.3819e+00,  5.5517e-02,  2.4470e+00],
         [ 5.7277e-01, -2.9413e+00, -2.1231e-01,  6.9449e-01, -1.2193e+00,
           9.9532e-01,  3.7752e-01,  3.6431e-01,  1.4683e+00, -6.4916e-01,
           1.3118e+00, -1.0181e+00, -5.9784e-01,  1.2726e+00,  1.4562e+00,
          -1.7664e+00,  1.3733e+00,  2.0294e+00,  1.3730e+00],
         [-1.2590e+00, -3.8298e+00, -1.3101e+00, -3.8968e-02, -7.7036e-01,
           1.1756e+00,  9.3893e-01,  1.5912e+00, -9.0418e-01, -4.7946e-01,
           1.7010e-02, -3.9544e-01, -5.8232e-01,  2.3650e+00,  1.7907e+00,
          -8.1933e-01,  1.1800e+00,  2.7833e+00,  1.9592e+00],
         [-9.5710e-01, -2.8819e+00, -7.7331e-01,  1.3686e-01, -3.1608e-01,
           6.8239e-01,  6.1304e-01,  6.3968e-01,  7.9641e-01, -4.8734e-01,
           5.0130e-02, -5.0114e-01, -1.6790e+00,  1.0732e+00,  5.4674e-01,
           5.0731e-01,  1.1297e+00,  9.5796e-01,  2.5914e+00],
         [-4.0961e-01, -3.9635e+00, -1.1470e+00,  5.1349e-01, -6.2367e-01,
           1.3788e+00,  1.1346e+00,  1.0005e+00,  9.5173e-01, -7.7085e-01,
           1.1279e+00, -8.1253e-01, -9.3141e-01,  2.0427e+00,  1.1967e+00,
          -1.2706e+00,  1.1265e+00,  3.1326e+00,  1.7260e+00]],
        grad_fn=<AddBackward0>)
--------------------------------------------------------------------------------
Epoch: 0001 cost = 3.849265
100%|██████████████████████████████████████████████████████████████████████████| 2/2 [00:00<00:00, 29.05it/s]

 ================================================================================
학습 완료 했으니, 전체 입력 데이터에 대해, prediction으로 학습 결과 확인

*** hidden_vectors[0]: tensor([[[-0.3743, -0.5019, -0.3546, -0.1557,  0.2206],
         [-0.3743, -0.5019, -0.3546, -0.1557,  0.2206],
         [-0.3743, -0.5019, -0.3546, -0.1557,  0.2206],
         [-0.0969, -0.4004, -0.0907, -0.0443,  0.0508],
         [-0.0969, -0.4004, -0.0907, -0.0443,  0.0508],
         [-0.0969, -0.4004, -0.0907, -0.0443,  0.0508],
         [-0.0969, -0.4004, -0.0907, -0.0443,  0.0508],
         [ 0.2534,  0.0280, -0.3516,  0.4409, -0.1869],
         [ 0.2534,  0.0280, -0.3516,  0.4409, -0.1869],
         [ 0.2534,  0.0280, -0.3516,  0.4409, -0.1869]],

        [[ 0.0692, -0.2882, -0.1287,  0.3104,  0.2757],
         [-0.4634, -0.2505, -0.2277,  0.1674, -0.2283],
         [-0.1169,  0.1250, -0.0807,  0.0135,  0.1884],
         [-0.4547, -0.0900,  0.5277,  0.0179,  0.3016],
         [ 0.1643, -0.3603, -0.2078,  0.3865,  0.1550],
         [-0.3843, -0.3244, -0.3034,  0.2502, -0.3444],
         [-0.0210,  0.0451, -0.1609,  0.0998,  0.0639],
         [-0.3291, -0.4530,  0.2644,  0.0948,  0.3435],
         [ 0.1915, -0.3562,  0.3029,  0.6700, -0.3653],
         [ 0.4430, -0.1001, -0.1456,  0.3030,  0.0137],
         [ 0.0999, -0.5632,  0.2436,  0.5185,  0.1933]]],
       grad_fn=<StackBackward0>)

*** hidden[0]: tensor([[[ 0.0692, -0.2882, -0.1287,  0.3104,  0.2757],
         [-0.4634, -0.2505, -0.2277,  0.1674, -0.2283],
         [-0.1169,  0.1250, -0.0807,  0.0135,  0.1884],
         [-0.4547, -0.0900,  0.5277,  0.0179,  0.3016],
         [ 0.1643, -0.3603, -0.2078,  0.3865,  0.1550],
         [-0.3843, -0.3244, -0.3034,  0.2502, -0.3444],
```

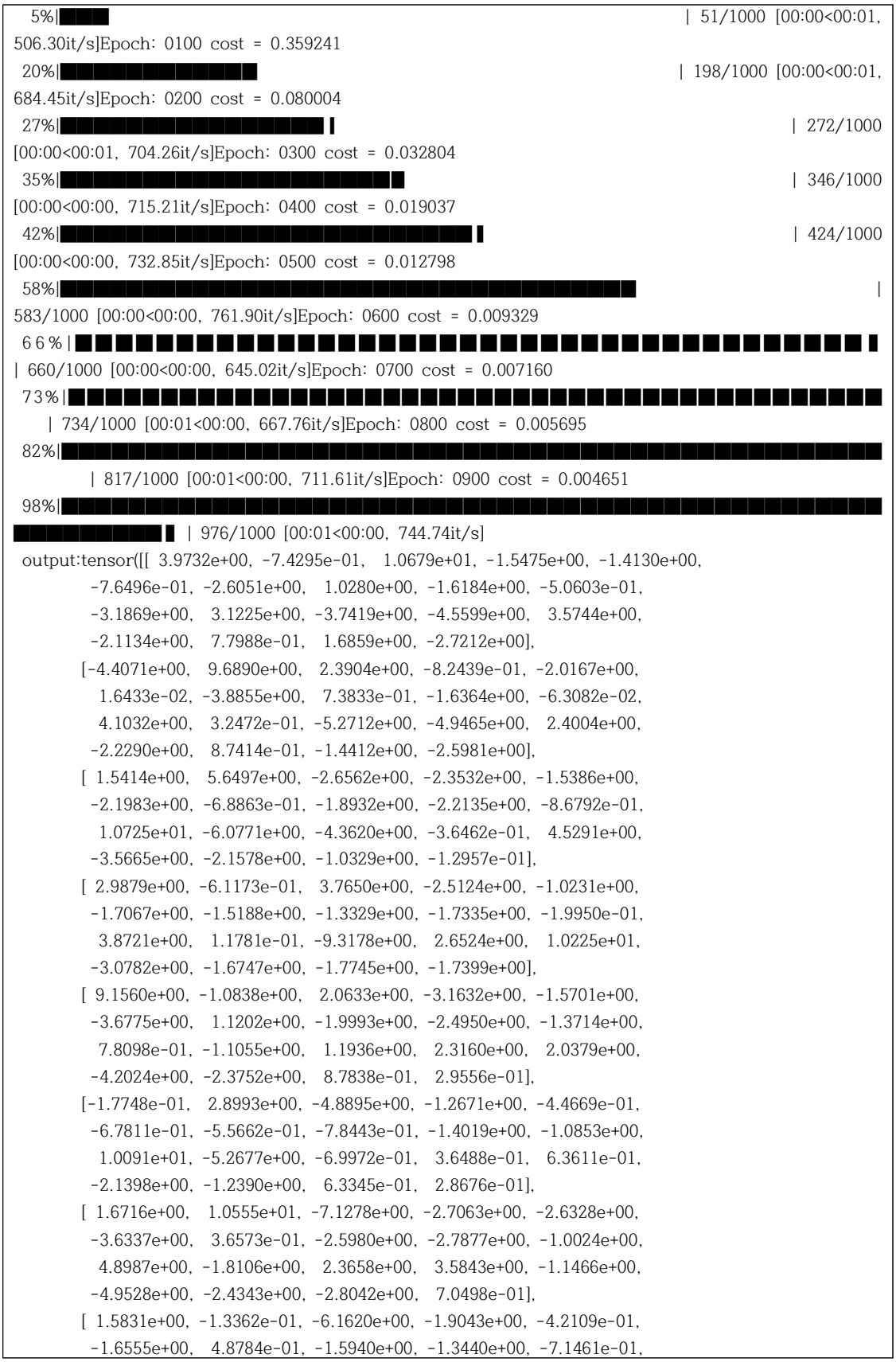
```
            [-0.0210,  0.0451, -0.1609,  0.0998,  0.0639],
            [-0.3291, -0.4530,  0.2644,  0.0948,  0.3435],
            [ 0.1915, -0.3562,  0.3029,  0.6700, -0.3653],
            [ 0.4430, -0.1001, -0.1456,  0.3030,  0.0137],
            [ 0.0999, -0.5632,  0.2436,  0.5185,  0.1933]]],
          grad_fn=<StackBackward0>)

 ..........................................................................
[학습후 테스트 결과]

[['i', 'like'], ['i', 'love'], ['i', 'hate'], ['i', 'enjoy'], ['you', 'like'], ['you', 'love'], ['you', 'hate'], ['you', 'want'], ['he', 'expects'], ['he', 'wants'], ['he', 'waits']] -> ['you', 'want', 'you', 'wants', 'you', 'want', 'you', 'wants', 'wants', 'you', 'wants']

(word2vec) D:\myprojects\nlp\rnn>
```

[3] 실행 결과 화면 (epoch = 1000인 경우의 학습 후)



```
  5%|████                                                                | 51/1000 [00:00<00:01,
506.30it/s]Epoch: 0100 cost = 0.359241
 20%|████████████████                                                   | 198/1000 [00:00<00:01,
684.45it/s]Epoch: 0200 cost = 0.080004
 27%|██████████████████████▏                                            | 272/1000
[00:00<00:01, 704.26it/s]Epoch: 0300 cost = 0.032804
 35%|████████████████████████████▏                                      | 346/1000
[00:00<00:00, 715.21it/s]Epoch: 0400 cost = 0.019037
 42%|██████████████████████████████████▏                                | 424/1000
[00:00<00:00, 732.85it/s]Epoch: 0500 cost = 0.012798
 58%|██████████████████████████████████████████████▍                    |
583/1000 [00:00<00:00, 761.90it/s]Epoch: 0600 cost = 0.009329
 66%|████████████████████████████████████████████████████▎              |
| 660/1000 [00:00<00:00, 645.02it/s]Epoch: 0700 cost = 0.007160
 73%|██████████████████████████████████████████████████████████▏        |
    | 734/1000 [00:01<00:00, 667.76it/s]Epoch: 0800 cost = 0.005695
 82%|████████████████████████████████████████████████████████████████▍  |
         | 817/1000 [00:01<00:00, 711.61it/s]Epoch: 0900 cost = 0.004651
 98%|████████████████████████████████████████████████████████████████▏  |
| 976/1000 [00:01<00:00, 744.74it/s]
output:tensor([[ 3.9732e+00, -7.4295e-01,  1.0679e+01, -1.5475e+00, -1.4130e+00,
         -7.6496e-01, -2.6051e+00,  1.0280e+00, -1.6184e+00, -5.0603e-01,
         -3.1869e+00,  3.1225e+00, -3.7419e+00, -4.5599e+00,  3.5744e+00,
         -2.1134e+00,  7.7988e-01,  1.6859e+00, -2.7212e+00],
        [-4.4071e+00,  9.6890e+00,  2.3904e+00, -8.2439e-01, -2.0167e+00,
          1.6433e-02, -3.8855e+00,  7.3833e-01, -1.6364e+00, -6.3082e-02,
          4.1032e+00,  3.2472e-01, -5.2712e+00, -4.9465e+00,  2.4004e+00,
         -2.2290e+00,  8.7414e-01, -1.4412e+00, -2.5981e+00],
        [ 1.5414e+00,  5.6497e+00, -2.6562e+00, -2.3532e+00, -1.5386e+00,
         -2.1983e+00, -6.8863e-01, -1.8932e+00, -2.2135e+00, -8.6792e-01,
          1.0725e+01, -6.0771e+00, -4.3620e+00, -3.6462e-01,  4.5291e+00,
         -3.5665e+00, -2.1578e+00, -1.0329e+00, -1.2957e-01],
        [ 2.9879e+00, -6.1173e-01,  3.7650e+00, -2.5124e+00, -1.0231e+00,
         -1.7067e+00, -1.5188e+00, -1.3329e+00, -1.7335e+00, -1.9950e-01,
          3.8721e+00,  1.1781e-01, -9.3178e+00,  2.6524e+00,  1.0225e+01,
         -3.0782e+00, -1.6747e+00, -1.7745e+00, -1.7399e+00],
        [ 9.1560e+00, -1.0838e+00,  2.0633e+00, -3.1632e+00, -1.5701e+00,
         -3.6775e+00,  1.1202e+00, -1.9993e+00, -2.4950e+00, -1.3714e+00,
          7.8098e-01, -1.1055e+00,  1.1936e+00,  2.3160e+00,  2.0379e+00,
         -4.2024e+00, -2.3752e+00,  8.7838e-01,  2.9556e-01],
        [-1.7748e-01,  2.8993e+00, -4.8895e+00, -1.2671e+00, -4.4669e-01,
         -6.7811e-01, -5.5662e-01, -7.8443e-01, -1.4019e+00, -1.0853e+00,
          1.0091e+01, -5.2677e+00, -6.9972e-01,  3.6488e-01,  6.3611e-01,
         -2.1398e+00, -1.2390e+00,  6.3345e-01,  2.8676e-01],
        [ 1.6716e+00,  1.0555e+01, -7.1278e+00, -2.7063e+00, -2.6328e+00,
         -3.6337e+00,  3.6573e-01, -2.5980e+00, -2.7877e+00, -1.0024e+00,
          4.8987e+00, -1.8106e+00,  2.3658e+00,  3.5843e+00, -1.1466e+00,
         -4.9528e+00, -2.4343e+00, -2.8042e+00,  7.0498e-01],
        [ 1.5831e+00, -1.3362e-01, -6.1620e+00, -1.9043e+00, -4.2109e-01,
         -1.6555e+00,  4.8784e-01, -1.5940e+00, -1.3440e+00, -7.1461e-01,
```

```
            1.1559e+00,  2.7416e+00,  6.1064e-01,  9.5525e+00,  1.6986e+00,
           -2.9542e+00, -1.8912e+00, -1.9064e+00,  3.3447e-01],
          [ 6.4686e-01,  2.3411e+00, -5.3618e+00, -4.1216e-01, -5.1260e-01,
           -3.5726e-01,  1.2945e-01,  6.5325e-01, -1.1409e+00, -1.5355e+00,
           -3.7832e-01,  1.3510e+00,  1.0276e+01,  1.4321e+00, -9.5501e+00,
           -1.6775e+00,  3.7272e-01,  2.5168e+00,  7.3887e-01],
          [-1.7582e+00, -9.9419e-03,  1.6405e+00, -2.1629e-01, -4.9273e-01,
            7.5057e-01, -2.2772e+00,  1.6833e+00, -5.8559e-01, -2.4122e-01,
           -8.3152e+00,  9.9620e+00,  2.3384e+00,  3.9064e+00, -2.3832e+00,
           -1.1171e+00,  1.6102e+00, -1.6989e-01, -2.0377e+00],
          [ 1.4766e+00, -2.3942e+00, -5.7177e+00, -1.5569e+00,  8.1021e-02,
           -1.0472e+00,  4.5251e-01, -1.1019e+00, -9.8438e-01, -7.2816e-01,
            4.9863e-02,  3.7017e+00,  1.1693e+00,  1.0020e+01,  1.2132e+00,
           -2.3450e+00, -1.5043e+00, -1.2159e+00,  2.6166e-01]],
         grad_fn=<AddBackward0>)
Epoch: 1000 cost = 0.003875
100%|██████████████████████████████████████████████████████████████████████████
██████████████| 1000/1000 [00:01<00:00, 701.66it/s]

 ================================================================================
[['i', 'like'], ['i', 'love'], ['i', 'hate'], ['i', 'enjoy'], ['you', 'like'], ['you', 'love'], ['you', 'hate'], ['you',
'want'], ['he', 'expects'], ['he', 'wants'], ['he', 'waits']] -> ['dogs', 'coffee', 'milk', 'tea', 'cats', 'milk',
'coffee', 'snowing', 'promotion', 'money', 'snowing']

(word2vec) D:\myprojects\nlp\rnn>
```

[4] 실행 결과 화면 (Prediction 의 예)

```
(word2vec) D:\myprojects\nlp\rnn>python rnn_m21.py you expects

 ================================================================================
word dictionary: {'cats': 0, 'coffee': 1, 'dogs': 2, 'enjoy': 3, 'expects': 4, 'hate': 5, 'he': 6, 'i': 7, 'like': 8, 'love': 9,
'milk': 10, 'money': 11, 'promotion': 12, 'snowing': 13, 'tea': 14, 'waits': 15, 'want': 16, 'wants': 17, 'you': 18}


 ================================================================================
학습용 input batch: tensor([[[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.],
         [0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.],
         [0., 0., 0., 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., 1.],
         [0., 0., 0., 0., 0., 0., 0., 0., 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., 1.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 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., 1.],
         [0., 0., 0., 0., 0., 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., 1.],
         [0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 0.,
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.],
         [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
          0., 0.]],

        [[0., 0., 0., 0., 0., 0., 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.,
          1., 0.]],

        [[0., 0., 0., 0., 0., 0., 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., 1., 0.,
          0., 0.]]], requires_grad=True)

 ================================================================================
학습용 target batch: tensor([ 2,  1, 10, 14,  0, 10,  1, 13, 12, 11, 13])

 ================================================================================

Usage [학습]: python rnn_m21.py train 100
Usage [Prediction]: python rnn_m21.py word1 word2
```

```
 --------------------------------------------------------------------------------
*** Prediction용 input batch:
 tensor([[[0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 1.],
          [0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
           0., 0.]]])

                                                                  <--- sequence 길이가 2 이므로
*** hidden_vectors[0]: tensor([[[ 0.6937,  0.9746, -0.2059,  0.7871,  0.6672]],   <--- 끝에서 두 번째 hidden vector
                               [[-0.6732,  0.8177, -0.9851,  0.9908, -0.8579]]],  <--- 마지막번째 hidden vector
        grad_fn=<StackBackward0>)

*** hidden[0]: tensor([[[-0.6732,  0.8177, -0.9851,  0.9908, -0.8579]]],
        grad_fn=<StackBackward0>)

*** Softmax Results: tensor([[1.5386e-03, 5.9495e-03, 2.0055e-02, 2.1666e-04, 9.3051e-06, 1.1651e-04,
          2.2295e-05, 8.6550e-05, 1.2712e-04, 1.8810e-06, 6.6586e-02, 1.6248e-06,
          9.0000e-01, 7.1958e-04, 5.2887e-07, 9.9300e-04, 3.2143e-06, 2.8805e-04,
          3.2852e-03]], grad_fn=<SoftmaxBackward0>)

 --------------------------------------------------------------------------------
전체 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']


 --------------------------------------------------------------------------------
predict 결과:tensor([[12]])
*** you  expects ==> promotion

(word2vec) D:\myprojects\nlp\rnn>
```

※ [Questions]

1. One2one, Many2one, Many2Many 등, RNN 구조가 다를 때의 차이를 명확히 이해하는가?

2 [Prediction 실행]. "python rnn_m21.py you want' 와 같이 단어 개수 2의 다양한 문장으로 rnn.py 프로그램을
실행해 보시오. 어떤 경우에 문제가 발생하는지와 문제 발생의 이유를 설명하시오.

3. [학습 데이터 추가후 실행] hidden layer의 벡터 길이를 10으로 변경하고, 학습 데이터 문장들을 10개 더 추가하여 실행하여 보시오. 이때 추가되는 문장에는 새로운 단어가 몇 개 더 들어가도록 하라

4. [batch_size 관련 코드 수정] 현재의 코드를 그대로 사용하면 입력 문장이 늘어나는 것에 비례하여 batch_size가 증가한다.
batch_size를 min(학습 문장의 총 개수, 32) 값으로 변경할 경우, 프로그램을 수정해 보시오

※ [실습 문제] - many(3)-to-one RNN Language Model

5. 소스코드에서 입력 단어의 길이를 3으로 변경하되, 학습 문장의 길이가 임의의 길이인 경우에 대하여 동작하도록 코드를 수정하시오. (각 문장에서 존재할 수 있는 단어의 개수가 임의의 개수이므로, sentence들로 부터 여러 개의 학습 window(window 크기는 크기 4 (input 3 + target 1))를 구하여 학습하여야 한다 cbow.py에서처럼..... 또, window크기 4 보다 작은 문장들은 학습 데이터에서 걸러내야 한다)