cbow

# example code

|  |
| --- |
| ## no library  def bow(sentence):    #(1) 입력받은 문장을 단어 단위로 쪼갠 뒤, 중복을 제거해�줍니다.    word\_list = sentence.split(' ')    word\_list = list(set(word\_list))    #(2) 단어의 수만큼 배열을 만들고, 0으로 채워�줍니다.    embedding\_matrix = [0 for element in range(len(word\_list))]    #(3) 각 인덱스의 단어가 몇 번 나오는지 count한뒤, 갱신해�줍니다.    for index, word in enumerate(word\_list):      embedding\_matrix[index] = sentence.count(word)    return word\_list, embedding\_matrix  sentence = "Suzy is very very pretty woman and YoonA is very pretty woman too"  word\_list, bow\_embedding = bow(sentence)  print("word\_list : ",word\_list,", embedding : ",bow\_embedding)  ## using sklearn  from sklearn.feature\_extraction.text import CountVectorizer  sentence = ["Suzy is very very pretty woman and YoonA is very pretty woman too"]  vectorizer = CountVectorizer(min\_df = 1, ngram\_range = (1,1))  embedding = vectorizer.fit\_transform(sentence)  vocab = vectorizer.get\_feature\_names()  print("word\_list : ",vocab,", embedding : ",embedding.toarray())  # convert context to index vector  def make\_context\_vector(context, word\_to\_ix):    idxs = [word\_to\_ix[w] for w in context]    return torch.tensor(idxs, dtype=torch.long)  # make dataset function  def make\_data(sentence):    data = []    for i in range(2, len(example\_sentence) - 2):      context = [example\_sentence[i - 2], example\_sentence[i - 1], example\_sentence[i + 1], example\_sentence[i + 2]]      target = example\_sentence[i]      data.append((context, target))    return data  #(4) CBOW 모델을 정의해 줍니다.  class CBOW(nn.Module):    def \_\_init\_\_(self, vocab\_size, embedding\_dim):      super(CBOW, self).\_\_init\_\_()      self.embeddings = nn.Embedding(vocab\_size, embedding\_dim)      self.layer1 = nn.Linear(embedding\_dim, 64)      self.activation1 = nn.ReLU()      self.layer2 = nn.Linear(64, vocab\_size)      self.activation2 = nn.LogSoftmax(dim = -1)    def forward(self, inputs):      embeded\_vector = sum(self.embeddings(inputs)).view(1,-1)      output = self.activation1(self.layer1(embeded\_vector))      output = self.activation2(self.layer2(output))      return output  ## using pytorch  import torch  import torch.nn as nn  EMBEDDING\_DIM = 128  EPOCHS = 100  with open('../[01]data\_set/data\_set.txt', 'r') as file:      example\_sentence = file.readline()  example\_sentence = example\_sentence.split()  print(example\_sentence)  #(1) 입력받은 문장을 단어로 쪼개고, 중복을 제거해줍니다.  vocab = set(example\_sentence)  vocab\_size = len(example\_sentence)  #(2) 단어 : 인덱스, 인덱스 : 단어를 가지는 딕셔너리를 선언해 줍니다.  word\_to\_index = {word:index for index, word in enumerate(vocab)}  index\_to\_word = {index:word for index, word in enumerate(vocab)}  #(3) 학습을 위한 데이터를 생성해 줍니다.  data = make\_data(example\_sentence)    #(5) 모델을 선언해주고, loss function, optimizer등을 선언해줍니다.  model = CBOW(vocab\_size, EMBEDDING\_DIM)  loss\_function = nn.NLLLoss()  optimizer = torch.optim.SGD(model.parameters(), lr=0.001)  #(6) 학습을 진행합니다.  for epoch in range(EPOCHS):      total\_loss = 0      for context, target in data:          context\_vector = make\_context\_vector(context, word\_to\_index)          log\_probs = model(context\_vector)          total\_loss += loss\_function(log\_probs, torch.tensor([word\_to\_index[target]]))      print('epoch = ',epoch, ', loss = ',total\_loss)      optimizer.zero\_grad()      total\_loss.backward()      optimizer.step()  #(7) test하고 싶은 문장을 뽑고, test를 진행합니다.  test\_data = ['CBOW','and','are','different.']  test\_vector = make\_context\_vector(test\_data, word\_to\_index)  result = model(test\_vector)  print('Prediction : ', index\_to\_word[torch.argmax(result[0]).item()]) |

# testing result

|  |
| --- |
| ['In', 'the', 'case', 'of', 'CBOW,', 'one', 'word', 'is', 'eliminated,', 'and', 'the', 'word', 'is', 'predicted', 'from', 'surrounding', 'words.', 'Therefore,', 'it', 'takes', 'multiple', 'input', 'vectors', 'as', 'inputs', 'to', 'the', 'model', 'and', 'creates', 'one', 'output', 'vector.', 'In', 'contrast,', 'Skip-Gram', 'learns', 'by', 'removing', 'all', 'words', 'except', 'one', 'word', 'and', 'predicting', 'the', 'surrounding', 'words', 'in', 'the', 'context', 'through', 'one', 'word.', 'So,', 'it', 'takes', 'a', 'vector', 'as', 'input', 'and', 'produces', 'multiple', 'output', 'vectors.', 'CBOW', 'and', 'Skip-Gram', 'are', 'different.']  epoch = 0 , loss = tensor(296.6661, grad\_fn=<AddBackward0>)  epoch = 1 , loss = tensor(287.5806, grad\_fn=<AddBackward0>)  epoch = 2 , loss = tensor(278.9847, grad\_fn=<AddBackward0>)  epoch = 3 , loss = tensor(270.7304, grad\_fn=<AddBackward0>)  epoch = 4 , loss = tensor(262.7270, grad\_fn=<AddBackward0>)  epoch = 5 , loss = tensor(254.9600, grad\_fn=<AddBackward0>)  epoch = 6 , loss = tensor(247.3369, grad\_fn=<AddBackward0>)  epoch = 7 , loss = tensor(239.7716, grad\_fn=<AddBackward0>)  epoch = 8 , loss = tensor(232.2176, grad\_fn=<AddBackward0>)  epoch = 9 , loss = tensor(224.5888, grad\_fn=<AddBackward0>)  epoch = 10 , loss = tensor(216.9380, grad\_fn=<AddBackward0>)  epoch = 11 , loss = tensor(209.3789, grad\_fn=<AddBackward0>)  epoch = 12 , loss = tensor(201.9030, grad\_fn=<AddBackward0>)  epoch = 13 , loss = tensor(194.5625, grad\_fn=<AddBackward0>)  epoch = 14 , loss = tensor(187.3731, grad\_fn=<AddBackward0>)  epoch = 15 , loss = tensor(180.3170, grad\_fn=<AddBackward0>)  epoch = 16 , loss = tensor(173.4122, grad\_fn=<AddBackward0>)  epoch = 17 , loss = tensor(166.5878, grad\_fn=<AddBackward0>)  epoch = 18 , loss = tensor(159.9040, grad\_fn=<AddBackward0>)  epoch = 19 , loss = tensor(153.3012, grad\_fn=<AddBackward0>)  epoch = 20 , loss = tensor(146.8092, grad\_fn=<AddBackward0>)  epoch = 21 , loss = tensor(140.3562, grad\_fn=<AddBackward0>)  epoch = 22 , loss = tensor(134.0317, grad\_fn=<AddBackward0>)  epoch = 23 , loss = tensor(127.8162, grad\_fn=<AddBackward0>)  epoch = 96 , loss = tensor(6.6704, grad\_fn=<AddBackward0>)  epoch = 97 , loss = tensor(6.5437, grad\_fn=<AddBackward0>)  epoch = 98 , loss = tensor(6.4217, grad\_fn=<AddBackward0>)  epoch = 99 , loss = tensor(6.3031, grad\_fn=<AddBackward0>)  Prediction : Skip-Gram |