|  |  |
| --- | --- |
| 교육제목 | 데이터 기반 인공지능 시스템 엔지니어 양성 과정 |
| 교육일시 | 2021-12-08 |
| 교육장소 | YGL 학과장 및 자택(디스코드 사용 온라인 학습) |
| **교육내용** | |
| 1. Bahdanau Attention     import tensorflow as tf  class BahdanauAttention(tf.keras.layers.Layer):  def \_\_init\_\_(self, units):  super(BahdanauAttention, self).\_\_init\_\_()  self.W\_decoder = tf.keras.layers.Dense(units)  self.W\_encoder = tf.keras.layers.Dense(units)  self.W\_combine = tf.keras.layers.Dense(1)    def call(self, H\_encoder, H\_decoder):  print("[H\_encoder shape :", H\_encoder.shape)  H\_encoder = self.W\_encoder(H\_encoder)  print("[W\_encoder X H\_encoder shape :", H\_encoder.shape)  print("\n[H\_decoder shape:", H\_decoder.shape)  H\_decoder = tf.expand\_dims(H\_decoder, 1)  H\_decoder = self.W\_decoder(H\_decoder)  print("[W\_decoder X H\_decoder] shape :", H\_decoder.shape)  score = self.W\_combine(tf.nn.tanh(H\_decoder+H\_encoder))  print("[Score Alignment] shape :", score.shape)  attention\_weights = tf.nn.softmax(score, axis= 1)  print("\n 최종 weight : \n", attention\_weights.numpy())  context\_vector = attention\_weights \* H\_decoder  context\_vector = tf.reduce\_sum(context\_vector, axis=1)  return context\_vector, attention\_weights  W\_size = 100  print("Hidden State를 {0}차원으로 Mapping\n".format(W\_size))  attention = BahdanauAttention(W\_size)  enc\_state = tf.random.uniform((1, 10, 512))  dec\_state = tf.random.uniform((1, 512))  \_ = attention(enc\_state, dec\_state) #def call(self, H\_encoder, H\_decoder):   1. Loung Attention     class LuongAttention(tf.keras.layers.Layer):  def \_\_init\_\_(self, units):  super(LuongAttention, self).\_\_init\_\_()  self.W\_combine = tf.keras.layers.Dense(units)  def call(self, H\_encoder, H\_decoder):  print("[H\_encoder] shape: ", H\_encoder.shape)  WH = self.W\_combine(H\_encoder)  print("[W\_encoder X H\_encoder] shape :", WH.shape)  H\_decoder = tf.expand\_dims(H\_decoder, 1)  alignment = tf.matmul(WH, tf.transpose(H\_decoder, [0, 2, 1]))  print("[Score\_alignmnet] Shape :", alignment.shape)  attention\_weights = tf.nn.softmax(alignment, axis=1)  print("\n 최종 weight : \n", attention\_weights.numpy())  attention\_weights = tf.squeeze(attention\_weights, axis=-1)  context\_vector = tf.matmul(attention\_weights, H\_encoder)  return context\_vector, attention\_weights  emb\_dim = 512  attention = LuongAttention(emb\_dim)  enc\_state = tf.random.uniform((1, 10, emb\_dim))  dec\_state = tf.random.uniform((1, emb\_dim))    \_ = attention(enc\_state, dec\_state) # def call(self, H\_encoder, H\_decoder):   1. Softmax   import numpy as np  import matplotlib.pyplot as plt  def softmax(x):  e\_x = np.exp(x - np.max(x))  return e\_x / e\_x.sum()  predicted\_logit = np.array([-2, 1.5, -1, 0.5, 2])  predicted\_prob = softmax(predicted\_logit)  plt.figure(figsize=[18,3])  plt.subplot(1, 2, 1)  plt.title('predicted logit')  plt.bar(np.arange(len(predicted\_logit)), predicted\_logit)  plt.grid()  plt.subplot(1, 2, 2)  plt.title('predicted probility')  plt.bar(np.arange(len(predicted\_prob)), predicted\_prob)  plt.grid()  plt.show()   1. Cross-entropy     - t: one-hot vector  - y probabilty  - k : 카테고리의 수  def cross\_entropy(t, y):  eps = 1e-8  ce = -np.sum(t \* np.log(y + eps))  return ce  predicted\_logit = np.array([-2, 1.5, -1, 0.5, 2])  predicted\_prob = softmax(predicted\_logit)  target\_prob = np.array([0, 0, 0, 0, 1])  plt.figure(figsize=[18,3])  plt.subplot(1,2,1)  plt.title('predicted probability')  plt.bar(np.arange(len(predicted\_prob)), predicted\_prob)  plt.grid()  plt.ylim([0,1])  plt.subplot(1, 2, 2)  plt.title('target\_probability')  plt.bar(np.arange(len(target\_prob)), target\_prob)  plt.grid()  plt.ylim([0, 1])  plt.show()  print('Cross-entropy :', cross\_entropy(target\_prob, predicted\_prob))  predicted\_logit = np.array([-2, 1.5, -1, 0.5, 4])  predicted\_prob = softmax(predicted\_logit)  target\_prob = np.array([0, 0, 0, 0, 1])  plt.figure(figsize=[18, 3])  plt.subplot(1, 2, 1)  plt.title('predicted probability')  plt.bar(np.arange(len(predicted\_prob)), predicted\_prob)  plt.grid()  plt.ylim([0, 1])  plt.subplot(1, 2, 2)  plt.title('target\_probability')  plt.bar(np.arange(len(target\_prob)), target\_prob)  plt.grid()  plt.ylim([0, 1])  plt.show()  print('Cross-entropy :', cross\_entropy(target\_prob, predicted\_prob))  predicted\_logit = np.array([-2, 1.5, -1, 0.5, 8])  predicted\_prob = softmax(predicted\_logit)  target\_prob = np.array([0, 0, 0, 0, 1])  plt.figure(figsize=[18, 3])  plt.subplot(1, 2, 1)  plt.title('predicted probability')  plt.bar(np.arange(len(predicted\_prob)), predicted\_prob)  plt.grid()  plt.ylim([0, 1])  plt.subplot(1, 2, 2)  plt.title('target\_probability')  plt.bar(np.arange(len(target\_prob)), target\_prob)  plt.grid()  plt.ylim([0, 1])  plt.show()  print('Cross-entropy :', cross\_entropy(target\_prob, predicted\_prob))   1. 양방향 LSTM + 어텐션 메커니즘 (IMDB 리뷰데이터)   from tensorflow.keras.datasets import imdb  from tensorflow.keras.utils import to\_categorical  from tensorflow.keras.preprocessing.sequence import pad\_sequences  vocab\_size = 10000  (x\_train, y\_train), (x\_test, y\_test) = imdb.load\_data(num\_words=vocab\_size)  print('리뷰의 최대 길이 : {}'.format(max(len(I)for I in x\_train)))  print('리뷰의 평균 길이 : {}'.format(sum(map(len, x\_train))/ len(x\_train)))  max\_len = 500  x\_train = pad\_sequences(x\_train, maxlen = max\_len)  x\_test = pad\_sequences(x\_test, maxlen = max\_len)  class BahdanauAttention(tf.keras.Model):  def \_\_init\_\_(self, units):  super(BahdanauAttention, self).\_\_init\_\_()  self.W1 = Dense(units)  self.W2 = Dense(units)  self.V = Dense(1)  def call(self, values, query):  # query size = (batch size, hidden size)  # hidden\_with\_time\_axis = (batch size, 1, hidden size)  hidden\_with\_time\_axis = tf.expand\_dims(query, 1)  # score = (batch size, max\_length, 1)  score = self.V(tf.nn.tanh(self.W1(values) + self.W2(hidden\_with\_time\_axis))) ###  # attention\_weights = (batch size, max\_length, 1)  attention\_weights = tf.nn.softmax(score, axis =1)  # context vector shape after sum = (batch size, hidden size)  context\_vector = attention\_weights \* values  context\_vector = tf.reduce\_sum(context\_vector, axis=1)  return context\_vector, attention\_weights  from tensorflow.keras.layers import Dense, Embedding, Bidirectional, LSTM, Concatenate, Dropout  from tensorflow.keras import Input, Model  from tensorflow.keras import optimizers  import os  sequence\_input = Input(shape = (max\_len, ), dtype = 'int32')  embedded\_sequences = Embedding(vocab\_size, 128, input\_length=max\_len, mask\_zero=True)(sequence\_input)  lstm = Bidirectional(LSTM(64, dropout = 0.5, return\_sequences = True))(embedded\_sequences)  lstm, forward\_h, forward\_c, backward\_h, backward\_c = Bidirectional(LSTM(64, dropout=0.5, return\_sequences=True, return\_state=True))(lstm)  print(lstm.shape, forward\_h.shape, forward\_c.shape, backward\_h.shape, backward\_c.shape)  state\_h = Concatenate()([forward\_h, backward\_h])  state\_c = Concatenate()([forward\_c, backward\_c])  attention = BahdanauAttention(64)  context\_vecotor, attention\_weights = attention(lstm, state\_h)  dense1 = Dense(20, activation='relu')(context\_vecotor)  dropout = Dropout(0.5)(dense1)  output = Dense(1, activation='sigmoid')(dropout)  model = Model(inputs= sequence\_input, outputs=output)  model.compile(loss='binary\_crossentropy', optimizer='adam', metrics=['accuracy'])  history = model.fit(x\_train, y\_train, epochs=20, batch\_size=256, validation\_data=(x\_test, y\_test), verbose=1) | |