|  |  |
| --- | --- |
| 교육제목 | 데이터 기반 인공지능 시스템 엔지니어 양성 과정 |
| 교육일시 | 2021-12-07 |
| 교육장소 | YGL 학과장 및 자택(디스코드 사용 온라인 학습) |
| **교육내용** | |
| 1. Class로 RNN 구현\\  * Tanh구현   import tensorflow as tf  from tensorflow.keras import layers, Model  import numpy as np  import matplotlib.pyplot as plt  class Tanh(Model):  def call(self, x):  return 2 / (1 + tf.exp(-2\*x)) – 1  x = tf.linspace(-5, 5, 1000)  y = Tanh()(x)  plt.figure(figsize=[8, 3])  plt.title('Tanh')  plt.scatter(x, y, s=1)  plt.grid()  plt.show()   * Dense Layer 구현   class Dense(Model):  def \_\_init\_\_(self, input\_dim, output\_dim):  super().\_\_init\_\_()  self.w = tf.random.normal(shape=(input\_dim, output\_dim))  self.b = tf.zeros(shape=(output\_dim))    def call(self, x):  # (batch, input\_dim)  # y = W\*x + b  y = x @ self.w + self.b  return y  t = tf.random.normal((2,256)) # x  t = Dense(256, 512)(t)  print(t.shape)   * RNNCell 구현   class RNNCell(Model):  def \_\_init\_\_(self, input\_dim, hidden\_dim, activation=Tanh):  super().\_\_init\_\_()  self.dense\_1 = Dense(hidden\_dim, hidden\_dim)  self.dense\_2 = Dense(input\_dim, hidden\_dim)  self.activation = activation()    def call(self, x, h):  # x : (batch, input\_dim)  h = self.activation(self.dense\_2(x)+self.dense\_1(h))  return h  x = tf.random.normal((2, 100, 16)) # input dim  h = tf.random.normal((2, 32)) # hidden dim  y = RNNCell(16, 32)(x,h)  print(y.shape)   * RNN   class RNN(Model):  def \_\_init\_\_(self, input\_dim, hidden\_dim, activation=Tanh):  super().\_\_init\_\_()  self.hidden\_dim = hidden\_dim  self.rnn\_cell = RNNCell(input\_dim, hidden\_dim, activation)    def call(self, x):  # x : (batch, length, input\_dim)  batch, length, input\_dim = x.shape  h = tf.zeros((batch, self.hidden\_dim)) # 초기 벡터를 0으로 초기화한다.  y = [] # 결과를 저장할 곳을 초기화 (빈 리스트)  for l in range(length): # timestep으로 loop  # batch, length, input\_dim  x\_ = x[:, l, :]  h = self.rnn\_cell(x\_, h)  y.append(h)  y = tf.stack(y, axis=1)  return y   * GRUCell 구현   class Sigmoid(Model):  def call(self, x):  return 1 / (1 + np.exp(-x))  x = tf.linspace(-5, 5, 1000)  y = Sigmoid()(x)  plt.figure(figsize=[8, 3])  plt.title('Sigmoid')  plt.scatter(x, y, s=1)  plt.grid()  plt.show()  class GRUCell(Model):  def \_\_init\_\_(self, input\_dim, hidden\_dim):  super().\_\_init\_\_()  self.sigmoid = Sigmoid()  self.tanh = Tanh()  self.dense\_ir = Dense(input\_dim, hidden\_dim)  self.dense\_hr = Dense(hidden\_dim, hidden\_dim)  self.dense\_iz = Dense(input\_dim, hidden\_dim)  self.dense\_hz = Dense(hidden\_dim, hidden\_dim)  self.dense\_in = Dense(input\_dim, hidden\_dim)  self.dense\_hn = Dense(hidden\_dim, hidden\_dim)  def call(self, x, h):  # x : (batch, input\_dim)  # h : (batch, hidden\_dim)    r = self.sigmoid(self.dense\_ir(x) + self.dense\_hr(h))  z = self.sigmoid(self.dense\_iz(x) + self.dense\_hz(h))  n = self.tanh(self.dense\_in(x) +r \* self.dense\_hn(h))  h = (1-z) \* n + z \* h  x = tf.random.normal((2, 16)) # input dim  h = tf.random.normal((2, 32)) # hidden dim  y = GRUCell(16, 32)(x,h)  print(y.shape)   * GRU 구현   class GRU(Model):  def \_\_init\_\_(self, input\_dim, hidden\_dim):  super().\_\_init\_\_()  self.hidden\_dim = hidden\_dim  self.gru\_cell = GRUCell(input\_dim, hidden\_dim)    def call(self, x):  # x : (batch, length, input\_dim)  batch, length, input\_dim = x.shape  h = tf.zeros((batch, self.hidden\_dim)) # 초기 벡터를 0으로 초기화한다.  y = [] # 결과를 저장할 곳을 초기화 (빈 리스트)  for l in range(length): # timestep으로 loop  # batch, length, input\_dim  x\_ = x[:, l, :]  h = self.gru\_cell(x\_, h)  y.append(h)  y = tf.stack(y, axis=1)  return y  x = tf.random.normal((2, 100, 16))  y = GRU(16, 32)(x)  print(y.shape)   * LSTM 구현   class LSTMCell(Model):  def \_\_init\_\_(self, input\_dim, hidden\_dim):  super().\_\_init\_\_()  self.sigmoid = Sigmoid()  self.tanh = Tanh()  self.dense = Dense(input\_dim + hidden\_dim, hidden\_dim\*4)    def call(self, x, state):  # x : (batch, input\_dim)  # state : (batch, hidden\_dim), (batch, hidden\_dim)  h, c = state  # (batch, input\_dim + hidden\_dim)  xh = np.concatenate([x, h], axis=1)  # (batch , hidden\_dim \*4)  d = self.dense(xh)  # (batch, hidden\_dim), (batch, hidden\_dim), (batch, hidden\_dim), (batch, hidden\_dim)  i, f, g, o = tf.split(d, 4, axis=1)  i = self.sigmoid(i)  f = self.sigmoid(f)  g = self.tanh(g)  o = self.sigmoid(o)  c\_ = f\*c + i\*g  h\_ = o\*self.tanh(c\_)  return h\_, c\_  class LSTM:  def \_\_init\_\_(self, input\_dim, hidden\_dim):  super().\_\_init\_\_()  self.hidden\_dim = hidden\_dim  self.lstm\_cell = LSTMCell(input\_dim, hidden\_dim)    def \_\_call\_\_(self, x):  # x : (batch, length, input\_dim)  batch, length, input\_dim = x.shape  h = tf.zeros((batch, self.hidden\_dim)) # 초기 벡터를 0으로 초기화한다.  c = tf.zeros((batch, self.hidden\_dim))  y = [] # 결과를 저장할 곳을 초기화 (빈 리스트)  for l in range(length): # timestep으로 loop  # batch, length, input\_dim  x\_ = x[:, l, :]  h, c = self.lstm\_cell(x\_, (h,c))  y.append(h)  y = tf.stack(y, axis=1)  return y  x = tf.random.normal((2, 100, 16))  y = LSTM(16, 32)(x)  print(y.shape)   1. RNN으로 소설 생성하기   import tensorflow as tf  import numpy as np  import os  import time  path\_to\_file = tf.keras.utils.get\_file('shakespeare.txt', 'https://storage.googleapis.com/download.tensorflow.org/data/shakespeare.txt')  print(path\_to\_file)  with open(path\_to\_file, 'r') as f:  lines = f.readlines()  for line in lines:  print(line.rstrip())  text = open(path\_to\_file, 'rb').read().decode(encoding='utf-8')  print('텍스트의 길이 : {}자'.format(len(text)))  vocab = sorted(set(text)) #중복제거  print('고유 문자 수 {}개'.format(len(vocab)))   * 텍스트 벡터화   for i, u in enumerate(vocab):  print(i, u)  for c in text[:100]:  print(c)  # 고유 문자에서 인덱스로 매핑 생성  char2idx = {u:i for i, u in enumerate(vocab)}  idx2char = np.array(vocab)  text\_as\_int = np.array([char2idx[c] for c in text])  print(len(text\_as\_int))  print(text\_as\_int[:50])  print('{')  for char,\_ in zip(char2idx, range(20)):  print(' {:4s}: {:3d},'.format(repr(char), char2idx[char]))  print(' ...\n')  # 텍스트에서 처음 13개의 문자가 숫자로 어떻게 매핑되었는지 봅시다.  print('{} ---- 문자들이 다음의 정수로 매핑되었습니다 ---- > {}'.format(repr(text[:13]), text\_as\_int[:13]))  len(text) % 101  # 단일 입력에 대해 원하는 문장의 최대 길이  seq\_length = 100  examples\_per\_epoch = len(text) // seq\_length  # 훈련 샘플/타킷 만들기  char\_dataset = tf.data.Dataset.from\_tensor\_slices(text\_as\_int) #텍스트 벡터 -> 문자 인덱스 스트림으로 변환  for i in char\_dataset.take(5):  print(idx2char[i.numpy()])  sequences = char\_dataset.batch(seq\_length+1, drop\_remainder=True)  # batch : 개별 문자들을 원하는 크기의 시퀀스로 쉽게 변환 할 수 있다.  for item in sequences.take(5):  print(repr(''.join(idx2char[item.numpy()])))  string = 'First Citizen'  print(string[:-1])  print(string[1:])  def split\_input\_target(chunk):  input\_text = chunk[0:-1]  target\_text = chunk[1:]  return input\_text, target\_text  dataset = sequences.map(split\_input\_target)  # map : 각 배치에 간단한 함수를 적용하고 입력 텍스트와 타깃 텍스트를 복사 및 이동합니다.  for input\_example, target\_example in dataset.take(1):  print('입력 데이터 :', repr(''.join(idx2char[input\_example.numpy()])))  print('타깃 데이터 :', repr(''.join(idx2char[target\_example.numpy()])))  for i, (input\_idx, target\_idx) in enumerate(zip(input\_example[:5], target\_example[:5])):  print("{:4d}단계".format(i))  print(" 입력: {} ({:s})".format(input\_idx, repr(idx2char[input\_idx])))  print(" 예상 출력: {} ({:s})".format(target\_idx, repr(idx2char[target\_idx])))   * 훈련 배치 생성   BATCH\_SIZE = 64  # 데이터셋을 섞을 버퍼 크기  BUFFER\_SIZE = 10000  dataset = dataset.shuffle(BUFFER\_SIZE).batch(BATCH\_SIZE, drop\_remainder=True)  dataset  class Embedding:  def \_\_init\_\_(self, input\_dim, output\_dim):  self.table = np.random.randn(input\_dim, output\_dim)  def \_\_call\_\_(self, x):  # x : (batch, length) int  batch, length = x.shape  x = x.flatten()  # (batch \* length. output\_dim)  y = self.table[x]  # (batch, length, output\_dim)  y = y.reshape(batch, length, -1)  return y  x = np.random.randint(0, len(vocab), size=(3, 100))  embedding = Embedding(len(vocab), 512)  y = embedding(x)  print(y.shape)  # 문자로 된 어휘 사전의 크기  vocab\_size = len(vocab)  print(vocab\_size)  # 임베딩 차원  embedding\_dim = 256  # RNN 갯수  rnn\_units = 1024  def build\_model(vocab\_size, embedding\_dim, rnn\_units, batch\_size):  model = tf.keras.Sequential([  tf.keras.layers.Embedding(vocab\_size, embedding\_dim, batch\_input\_shape=[batch\_size, None]),  tf.keras.layers.LSTM(rnn\_units, return\_sequences=True, stateful=True, recurrent\_initializer='glorot\_uniform'),  tf.keras.layers.Dense(vocab\_size)  ])  return model  model = build\_model(vocab\_size = len(vocab),  embedding\_dim = embedding\_dim,  rnn\_units = rnn\_units,  batch\_size=BATCH\_SIZE)   * 모델 사용   # 출력 형태 살펴보기  for input\_example\_batch, target\_example\_batch in dataset.take(1):  example\_batch\_predictions = model(input\_example\_batch)  print(example\_batch\_predictions.shape, "# (배치크기, 시퀀스 길이, 어휘 사전 크기)")  model.summary()  # 배치의 첫 번째 샘플링  example\_batch\_predictions[0].shape  sampled\_indices = tf.random.categorical(example\_batch\_predictions[0], num\_samples=1)  print(sampled\_indices.shape)  sampled\_indices= tf.squeeze(sampled\_indices, axis=-1).numpy()  print(sampled\_indices.shape)  print("입력 : \n", repr("".join(idx2char[input\_example\_batch[0]])))  print()  print("예측된 다음 문자: \n", repr("".join(idx2char[sampled\_indices])))  def loss(labels, logits):  return tf.keras.losses.sparse\_categorical\_crossentropy(labels, logits, from\_logits=True)  example\_batch\_loss = loss(target\_example\_batch, example\_batch\_predictions)  print("예측 배열 크기(shape) :", example\_batch\_predictions.shape)  print("스칼라 손실 :", example\_batch\_loss.numpy().mean())  model.compile(optimizer = 'adam', loss = loss)  # 체크포인트가 저장될 디렉토리  checkpoint\_dir = './training\_checkpoints'  # 체크포인트 파일 이름  checkpoint\_prefix = os.path.join(checkpoint\_dir, "ckpt\_{epoch}")  checkpoint\_callback=tf.keras.callbacks.ModelCheckpoint(  filepath=checkpoint\_prefix,  save\_weights\_only=True)  Epoch = 10  history = model.fit(dataset, epochs=Epoch, callbacks=[checkpoint\_callback])  tf.train.latest\_checkpoint(checkpoint\_dir)  model = build\_model(vocab\_size, embedding\_dim, rnn\_units, batch\_size=1)  model.load\_weights(tf.train.latest\_checkpoint(checkpoint\_dir))  model.build(tf.TensorShape([1, None]))  model.summary()  def generate\_text(model, start\_string):  # 평가 단계 (학습된 모델을 사용하여 텍스트 생성)  #생성할 문자의 수  num\_generate= 1000  # 시작 문자열을 숫자로 변환(벡터화)  input\_eval = [char2idx[s] for s in start\_string]  # (1 len(start\_string))  input\_eval = tf.expand\_dims(input\_eval, 0)  #결과를 저장할 빈 문자열  text\_generated= []  # 온도가 낮으면 더 에측 가능한 텍스트가 되고  # 온도가 높으면 더 의외의 텍스트가 됩니다.  # 최적의 세팅을 찾기 위한 실험  temperature = 1.0  # 여기에서 배치 크기 == 1  model.reset\_states()  for i in range(num\_generate):  predictions = model(input\_eval)  # 배치 차원 제거  predictions = tf.squeeze(predictions, 0)  #범주형 분포를 사용하여 모델에서 리턴한 단어 예측  predictions = predictions/ temperature  predicted\_id = tf.random.categorical(predictions, num\_samples=1)[-1,0].numpy()  # 예측된 단어를 다음 입력으로 모델에 전달  # 이전 은닉 상태와 함께  input\_eval = tf.expand\_dims([predicted\_id], 0)  text\_generated.append(idx2char[predicted\_id])  return (start\_string + ''.join(text\_generated))  print(generate\_text(model, start\_string=u"ROMEO: "))   1. Seq2Seq 모델   import tensorflow as tf  class Encoder(tf.keras.Model):  def \_\_init\_\_(self, vocab\_size, embedding\_dim, enc\_units):  super(Encoder, self).\_\_init\_\_()  self.embedding = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)  self.lstm = tf.keras.layers.LSTM(enc\_units)  def call(self, x):  print('입력 shape :', x.shape)  x = self.embedding(x)  print('Enbedding Layer 거친 shape :', x.shape)    output = self.lstm(x)  print('LSTM shape의 output shape :', output.shape)  return output  vocab\_size = 30000  emb\_size = 256  lstm\_size = 512  batch\_size = 1  sample\_seq\_len = 3  print("Vocab Size : {0}".format(vocab\_size))  print("Embedding Size : {0}".format(emb\_size))  print("LSTM Size : {0}".format(lstm\_size))  print("Batch Size : {0}".format(batch\_size))  print("Sample Sequence Length : {0}\n".format(sample\_seq\_len))  encoder = Encoder(vocab\_size, emb\_size, lstm\_size) # def \_\_init\_\_(self, vocab\_size, embedding\_dim, enc\_units):  sample\_input = tf.zeros((batch\_size, sample\_seq\_len))# 춤 추는 소시지// ~ embedding layer ~ ~lstm layer~  sample\_output = encoder(sample\_input) # def call(self, x):   1. LSTM Decoder   class Decoder(tf.keras.Model):  def \_\_init\_\_(self, vocab\_size, embedding\_dim, dec\_units):  super(Decoder, self).\_\_init\_\_()  self.embedding = tf.keras.layers.Embedding(vocab\_size, embedding\_dim)  self.lstm = tf.keras.layers.LSTM(dec\_units, return\_sequences= True)  self.fc = tf.keras.layers.Dense(vocab\_size)  self.softmax = tf.keras.layers.Softmax(axis=-1)  def call(self, x, context\_v):  print("입력 shape :", x.shape)  x = self.embedding(x)  print("Embedding Layer을 거친 shape :", x.shape)  context\_v = tf.repeat(tf.expand\_dims(context\_v, axis=1), repeats=x.shape[1], axis=1)  x = tf.concat([x, context\_v], axis= -1)  print("Context Vector가 더해진 shape :", x.shape)  x = self.lstm(x)  print("LSTM Layer의 Output layer:", x.shape)  output = self.fc(x)  print("Decoder의 최종 Output shape :", output.shape)  return self.softmax(output)  vocab\_size = 30000  emb\_size = 256  lstm\_size = 512  batch\_size = 1  sample\_seq\_len = 3  print("Vocab Size : {0}".format(vocab\_size))  print("Embedding Size : {0}".format(emb\_size))  print("LSTM Size : {0}".format(lstm\_size))  print("Batch Size : {0}".format(batch\_size))  print("Sample Sequence Length : {0}\n".format(sample\_seq\_len))  decoder = Decoder(vocab\_size, emb\_size, lstm\_size)  sample\_input = tf.zeros((batch\_size, sample\_seq\_len))  dec\_output = decoder(sample\_input, sample\_output) | |