|  |  |
| --- | --- |
| 교육제목 | 데이터 기반 인공지능 시스템 엔지니어 양성 과정 |
| 교육일시 | 211102 |
| 교육장소 | YGL 학과장 및 자택(디스코드 사용 온라인 학습) |
| **교육내용** | |
| 1. RNN(순환 신경망)   입력과 출력을 시퀀스 단위로 처리하는 모델. 번역기를 예로 들어, 입력은 번역하고자 하는 문장. 즉 단어 시퀀스임.  출력에 해당되는 번역된 문장 또한 단어 시퀀스.  이러한 시퀀스들을 처리하기 위해 고안된 모델들을 시퀀스 모델이라고 함. 그 중에서도 RNN은 딥 러닝에 있어 가장 기본적인 시퀀스 모델임.  LSTM이나 GRU 또한 근본적으로 RNN에 속함.  https://wikidocs.net/images/page/22886/rnn_image1_ver2.PNG   * x는 입력층의 입력 벡터, y는 출력층의 출력 벡터. 편향 b도 입력으로 존재할 수 있지만 생략됨. RNN에서 은닉층에서 활성화 함수를 통해 결과를 내보내는 역할을 하는 노드를 셀(cell)이라고 함. * 이 셀은 이전의 값을 기억하려고 하는 일종의 메모리 역할을 수행하므로 이를 메모리 셀 또는 RNN 셀이라고 표현   은닉층의 메모리 셀은 **각각의 시점(time step)에서 바로 이전 시점에서의 은닉층의 메모리 셀에서 나온 값을 자신의 입력으로 사용하는 재귀적 활동**을 하고 있음. t 시점의 메모리 셀은 t-1 시점의 메모리 셀이 보낸 은닉 상태값을 t 시점의 은닉 상태 계산을 위한 입력값으로 사용함.  https://wikidocs.net/images/page/22886/rnn_image3_ver2.PNG  RNN은 입력과 출력의 길이를 다르게 설계 할 수 있으므로 다양한 용도로 사용할 수 있음. 위 그림은 입력과 출력의 길이에 따라서 달라지는 RNN의 다양한 형태임.   * Keras로 RNN 구현하기   RNN 층을 추가하는 코드  model.add(SimpleRNN(hidden\_size)) #가장 간단한 형태  # 추가 인자를 사용할 때  model.add(SimpleRNN(hidden\_size, input\_shape = (timesteps, input\_dim))  # 다른 표기  model.add(SimpleRNN(hidden\_size, input\_length=M, input\_dim=N))  from tensorflow.keras.models import Sequential  from tensorflow.keras.layers import SimpleRNN  model = Sequential()  model.add(SimpleRNN(3, input\_shape=(2,10)))  # model.add(SimpleRNN(3, input\_length=2, input\_dim=10)) 동일  model.summary()  model= Sequential()  model.add(SimpleRNN(3, batch\_input\_shape=(8, 2, 10)))  model.summary()  model= Sequential()  model.add(SimpleRNN(3, batch\_input\_shape=(8, 2, 10), return\_sequences=True))  model.summary()   * 파이썬으로 RNN구현하기   ht=tanh(WxXt+Wh ht−1+b)  hidden\_state = 0 # 초기 은닉 상태를 0으로 초기화  for input\_t in input\_length: # 각 시점마다 입력을 받는다.  output\_t = tanh(input\_t, hidden\_state\_t) # 각 시점에 대해서 입력과 은닉 상태를 가지고 연산  hidden\_state\_t = output\_t # 계산 결과는 현재 시점의 은닉 상태가 된다.  import numpy as np  timesteps = 10  input\_dim = 4  hidden\_size = 8  # 입력에 해당되는 2D 텐서  inputs = np.random.random((timesteps, input\_dim))  # 초기 은닉 상태는 0벡터로 초기화  hidden\_state\_t = np.zeros((hidden\_size,))  print(hidden\_state\_t) # 8의 크기를 가지는 hidden state. 현재는 초기 hidden state로 모든 차원이 0의 값을 가짐  Wx = np.random.random((hidden\_size, input\_dim)) #(8,4) 2D텐서 생성 입력에 대한 가중치  Wh = np.random.random((hidden\_size, hidden\_size)) # (8,8)크기의 2D텐서 생성. hidden state에 대한 가중치  b = np.random.random((hidden\_size,)) #(8,)크기의 1D텐서 생성. 편향(bias)  print(np.shape(Wx)) # hidden state x 입력의 차원  print(np.shape(Wh)) # hidden state x hidden state size  print(np.shape(b)) # hidden state size  total\_hidden\_states= []  # 메모리 셀 동작  for input\_t in inputs: # 각 시점에 따라서 입력값이 입력이 됨.  output\_t = np.tanh(np.dot(Wx, input\_t) + np.dot(Wh, hidden\_state\_t)+b)  # Wx\* wt + Wh \* Ht-1 + b  total\_hidden\_states.append(list(output\_t)) #각 시점의 은닉 상태의 값을 계속해서 축적  print(np.shape(total\_hidden\_states)) # 각 시점 t별 메모리 셀의 출력의 크기는 (timestep, output\_dim)  hidden\_state\_t = output\_t  total\_hidden\_state = np.stack(total\_hidden\_states, axis=0)  # (timesteps, output\_dim)  print(total\_hidden\_states)   * 더 깊은 RNN   model = Sequential()  model.add(SimpleRNN(hidden\_size, input\_length =10, input\_dim =5, return\_sequences=True))  model.add(SimpleRNN(hidden\_size, return\_sequences = True))  model.summary()   * BiLSTM   from tensorflow.keras.layers import Bidirectional  timesteps = 10  input\_dim = 5  model = Sequential()  model.add(Bidirectional(SimpleRNN(hidden\_size, return\_sequences=True, input\_shape=(timesteps, input\_dim))))  model = Sequential()  model.add(Bidirectional(SimpleRNN(hidden\_size, return\_sequences=True, input\_shape=(timesteps, input\_dim))))  model.add(Bidirectional(SimpleRNN(hidden\_size, return\_sequences=True)))  model.add(Bidirectional(SimpleRNN(hidden\_size, return\_sequences=True)))  model.add(Bidirectional(SimpleRNN(hidden\_size, return\_sequences=True)))   * 임의의 입력으로 Simple RNN 생성   import numpy as np  import tensorflow as tf  from tensorflow.keras.layers import SimpleRNN, LSTM, Bidirectional  train\_x = [[0.1, 4.2, 1.5, 1.1, 2.8],  [1.0, 3.1, 2.5, 0.7, 1.1],  [0.3, 2.1, 1.5, 2.1, 0.1],  [2.2, 1.4, 0.5, 0.9, 1.1]]  print(np.shape(train\_x)) # 단어 벡터의 차원은 5, 문장의 길이 4  train\_x = [[[0.1, 4.2, 1.5, 1.1, 2.8],  [1.0, 3.1, 2.5, 0.7, 1.1],  [0.3, 2.1, 1.5, 2.1, 0.1],  [2.2, 1.4, 0.5, 0.9, 1.1]]]  train\_x = np.array(train\_x, dtype=np.float32)  print(train\_x.shape) # (batch\_size, timesteps, input\_dim)  rnn = SimpleRNN(3)  # rnn = SimpleRNN(3, return\_sequences=False, return\_state=False)와 동일  hidden\_state = rnn(train\_x)  print('hidden state : {}, shape: {}'.format(hidden\_state, hidden\_state.shape))  rnn = SimpleRNN(3, return\_sequences=True)  hidden\_states = rnn(train\_x)  print('hidden state : {}, shape : {}'.format(hidden\_states, hidden\_states.shape))  rnn = SimpleRNN(3, return\_sequences=True, return\_state=True)  hidden\_states, last\_states = rnn(train\_x)  print('hidden states : {}, shape: {}'.format(hidden\_states, hidden\_states.shape))  print('last hidden state : {}, shape : {}'.format(last\_states, last\_states.shape))  rnn = SimpleRNN(3, return\_sequences=False, return\_state=True)  hidden\_state, last\_state = rnn(train\_x)  print('hidden state : {}, shape : {}'.format(hidden\_state, hidden\_state.shape))  print('last hidden state : {}, shape : {}'.format(last\_state, last\_state.shape))   * LSTM 이해하기   lstm = LSTM(3, return\_sequences=False, return\_state=True)  hidden\_state, last\_state, last\_cell\_state = lstm(train\_x)  print('hidden state : {}, shape : {}'.format(hidden\_state, hidden\_state.shape))  print('last hidden state : {}, shape : {}'.format(last\_state, last\_state.shape))  print('last cell state : {}, shape : {}'.format(last\_cell\_state, last\_cell\_state.shape))  lstm = LSTM(3, return\_sequences= True, return\_state = True)  hidden\_states, last\_hidden\_state, last\_cell\_state = lstm(train\_x)  print('hidden states : {}, shape : {}'.format(hidden\_states, hidden\_states.shape))  print('last hidden state : {}, shape : {}'.format(last\_hidden\_state, last\_hidden\_state.shape))  print('last cell state : {}, shape : {}'.format(last\_cell\_state, last\_cell\_state.shape))   * LSTM 이해하기 2   x = np.array([[1,2, 3],  [2,3,4],  [3,4,5],  [4,5,6],  [5,6,7],  [6,7,8],  [7,8,9],  [8,9,10],  [9,10,11],  [10, 11, 12],  [20, 30, 40],  [30, 40, 50],  [40, 50, 60]])  y = np.array([4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 50, 60, 70])  x.shape, y.shape  x = x.reshape((x.shape[0], x.shape[1], 1))  print(x.shape)  import tensorflow as tf  from tensorflow import keras  model = keras.Sequential()  model.add(keras.layers.LSTM(20, activation='relu', input\_shape=(3,1)))  model.add(keras.layers.Dense(5, activation='relu'))  model.add(keras.layers.Dense(1))  model.compile(optimizer='adam', loss='mse')  es = keras.callbacks.EarlyStopping(monitor='loss', patience=10, mode='auto')  model.fit(x, y, epochs=1000, batch\_size=1, verbose=1, callbacks=[es])  x\_test = np.array([25, 35, 45])  x\_test = x\_test.reshape((1, 3, 1))  pred = model.predict(x\_test)  pred | |