transformers

# example code

|  |
| --- |
| import numpy as np  import matplotlib.pyplot as plt  import tensorflow as tf  class PositionalEncoding(tf.keras.layers.Layer):    def \_\_init\_\_(self, position, d\_model):      super(PositionalEncoding, self).\_\_init\_\_()      self.pos\_encoding = self.positional\_encoding(position, d\_model)    def get\_angles(self, position, i, d\_model):      angles = 1 / tf.pow(10000, (2 \* (i // 2)) / tf.cast(d\_model, tf.float32))      return position \* angles    def positional\_encoding(self, position, d\_model):      angle\_rads = self.get\_angles(          position=tf.range(position, dtype=tf.float32)[:, tf.newaxis],          i=tf.range(d\_model, dtype=tf.float32)[tf.newaxis, :],          d\_model=d\_model)      # 배열의 짝수 인덱스(2i)에는 사인 함수 적용      sines = tf.math.sin(angle\_rads[:, 0::2])      # 배열의 홀수 인덱스(2i+1)에는 코사인 함수 적용      cosines = tf.math.cos(angle\_rads[:, 1::2])      angle\_rads = np.zeros(angle\_rads.shape)      angle\_rads[:, 0::2] = sines      angle\_rads[:, 1::2] = cosines      pos\_encoding = tf.constant(angle\_rads)      pos\_encoding = pos\_encoding[tf.newaxis, ...]      print(pos\_encoding.shape)      return tf.cast(pos\_encoding, tf.float32)    def call(self, inputs):      return inputs + self.pos\_encoding[:, :tf.shape(inputs)[1], :]  # 문장의 길이 50, 임베딩 벡터의 차원 128  sample\_pos\_encoding = PositionalEncoding(50, 128)  plt.pcolormesh(sample\_pos\_encoding.pos\_encoding.numpy()[0], cmap='RdBu')  plt.xlabel('Depth')  plt.xlim((0, 128))  plt.ylabel('Position')  plt.colorbar()  plt.show()  def scaled\_dot\_product\_attention(query, key, value, mask):    # query 크기 : (batch\_size, num\_heads, query의 문장 길이, d\_model/num\_heads)    # key 크기 : (batch\_size, num\_heads, key의 문장 길이, d\_model/num\_heads)    # value 크기 : (batch\_size, num\_heads, value의 문장 길이, d\_model/num\_heads)    # padding\_mask : (batch\_size, 1, 1, key의 문장 길이)    # Q와 K의 곱. 어텐션 스코어 행렬.    matmul\_qk = tf.matmul(query, key, transpose\_b=True)    # 스케일링    # dk의 루트값으로 나눠준다.    depth = tf.cast(tf.shape(key)[-1], tf.float32)    logits = matmul\_qk / tf.math.sqrt(depth)    # 마스킹. 어텐션 스코어 행렬의 마스킹 할 위치에 매우 작은 음수값을 넣는다.    # 매우 작은 값이므로 소프트맥스 함수를 지나면 행렬의 해당 위치의 값은 0이 된다.    if mask is not None:      logits += (mask \* -1e9)    # 소프트맥스 함수는 마지막 차원인 key의 문장 길이 방향으로 수행된다.    # attention weight : (batch\_size, num\_heads, query의 문장 길이, key의 문장 길이)    attention\_weights = tf.nn.softmax(logits, axis=-1)    # output : (batch\_size, num\_heads, query의 문장 길이, d\_model/num\_heads)    output = tf.matmul(attention\_weights, value)    return output, attention\_weights  # 임의의 Query, Key, Value인 Q, K, V 행렬 생성  np.set\_printoptions(suppress=True)  temp\_k = tf.constant([[10,0,0],                        [0,10,0],                        [0,0,10],                        [0,0,10]], dtype=tf.float32)  # (4, 3)  temp\_v = tf.constant([[   1,0],                        [  10,0],                        [ 100,5],                        [1000,6]], dtype=tf.float32)  # (4, 2)  temp\_q = tf.constant([[0, 10, 0]], dtype=tf.float32)  # (1, 3)  temp\_q = tf.constant([[0, 0, 10]], dtype=tf.float32)  temp\_out, temp\_attn = scaled\_dot\_product\_attention(temp\_q, temp\_k, temp\_v, None)  print(temp\_attn) # 어텐션 분포(어텐션 가중치의 나열)  print(temp\_out) # 어텐션 값  temp\_q = tf.constant([[0, 0, 10], [0, 10, 0], [10, 10, 0]], dtype=tf.float32)  # (3, 3)  temp\_out, temp\_attn = scaled\_dot\_product\_attention(temp\_q, temp\_k, temp\_v, None)  print(temp\_attn) # 어텐션 분포(어텐션 가중치의 나열)  print(temp\_out) # 어텐션 값    class MultiHeadAttention(tf.keras.layers.Layer):    def \_\_init\_\_(self, d\_model, num\_heads, name="multi\_head\_attention"):      super(MultiHeadAttention, self).\_\_init\_\_(name=name)      self.num\_heads = num\_heads      self.d\_model = d\_model      assert d\_model % self.num\_heads == 0      # d\_model을 num\_heads로 나눈 값.      # 논문 기준 : 64      self.depth = d\_model // self.num\_heads      # WQ, WK, WV에 해당하는 밀집층 정의      self.query\_dense = tf.keras.layers.Dense(units=d\_model)      self.key\_dense = tf.keras.layers.Dense(units=d\_model)      self.value\_dense = tf.keras.layers.Dense(units=d\_model)      # WO에 해당하는 밀집층 정의      self.dense = tf.keras.layers.Dense(units=d\_model)    # num\_heads 개수만큼 q, k, v를 split하는 함수    def split\_heads(self, inputs, batch\_size):      inputs = tf.reshape(          inputs, shape=(batch\_size, -1, self.num\_heads, self.depth))      return tf.transpose(inputs, perm=[0, 2, 1, 3])    def call(self, inputs):      query, key, value, mask = inputs['query'], inputs['key'], inputs[          'value'], inputs['mask']      batch\_size = tf.shape(query)[0]      # 1. WQ, WK, WV에 해당하는 밀집층 지나기      # q : (batch\_size, query의 문장 길이, d\_model)      # k : (batch\_size, key의 문장 길이, d\_model)      # v : (batch\_size, value의 문장 길이, d\_model)      # 참고) 인코더(k, v)-디코더(q) 어텐션에서는 query 길이와 key, value의 길이는 다를 수 있다.      query = self.query\_dense(query)      key = self.key\_dense(key)      value = self.value\_dense(value)      # 2. 헤드 나누기      # q : (batch\_size, num\_heads, query의 문장 길이, d\_model/num\_heads)      # k : (batch\_size, num\_heads, key의 문장 길이, d\_model/num\_heads)      # v : (batch\_size, num\_heads, value의 문장 길이, d\_model/num\_heads)      query = self.split\_heads(query, batch\_size)      key = self.split\_heads(key, batch\_size)      value = self.split\_heads(value, batch\_size)      # 3. 스케일드 닷 프로덕트 어텐션. 앞서 구현한 함수 사용.      # (batch\_size, num\_heads, query의 문장 길이, d\_model/num\_heads)      scaled\_attention, \_ = scaled\_dot\_product\_attention(query, key, value, mask)      # (batch\_size, query의 문장 길이, num\_heads, d\_model/num\_heads)      scaled\_attention = tf.transpose(scaled\_attention, perm=[0, 2, 1, 3])      # 4. 헤드 연결(concatenate)하기      # (batch\_size, query의 문장 길이, d\_model)      concat\_attention = tf.reshape(scaled\_attention,                                    (batch\_size, -1, self.d\_model))      # 5. WO에 해당하는 밀집층 지나기      # (batch\_size, query의 문장 길이, d\_model)      outputs = self.dense(concat\_attention)      return outputs  def create\_padding\_mask(x):    mask = tf.cast(tf.math.equal(x, 0), tf.float32)    # (batch\_size, 1, 1, key의 문장 길이)    return mask[:, tf.newaxis, tf.newaxis, :]  def encoder\_layer(dff, d\_model, num\_heads, dropout, name="encoder\_layer"):    inputs = tf.keras.Input(shape=(None, d\_model), name="inputs")    # 인코더는 패딩 마스크 사용    padding\_mask = tf.keras.Input(shape=(1, 1, None), name="padding\_mask")    # 멀티-헤드 어텐션 (첫번째 서브층 / 셀프 어텐션)    attention = MultiHeadAttention(        d\_model, num\_heads, name="attention")({            'query': inputs, 'key': inputs, 'value': inputs, # Q = K = V            'mask': padding\_mask # 패딩 마스크 사용        })    # 드롭아웃 + 잔차 연결과 층 정규화    attention = tf.keras.layers.Dropout(rate=dropout)(attention)    attention = tf.keras.layers.LayerNormalization(        epsilon=1e-6)(inputs + attention)    # 포지션 와이즈 피드 포워드 신경망 (두번째 서브층)    outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention)    outputs = tf.keras.layers.Dense(units=d\_model)(outputs)    # 드롭아웃 + 잔차 연결과 층 정규화    outputs = tf.keras.layers.Dropout(rate=dropout)(outputs)    outputs = tf.keras.layers.LayerNormalization(        epsilon=1e-6)(attention + outputs)    return tf.keras.Model(        inputs=[inputs, padding\_mask], outputs=outputs, name=name)  def encoder(vocab\_size, num\_layers, dff,              d\_model, num\_heads, dropout,              name="encoder"):    inputs = tf.keras.Input(shape=(None,), name="inputs")    # 인코더는 패딩 마스크 사용    padding\_mask = tf.keras.Input(shape=(1, 1, None), name="padding\_mask")    # 포지셔널 인코딩 + 드롭아웃    embeddings = tf.keras.layers.Embedding(vocab\_size, d\_model)(inputs)    embeddings \*= tf.math.sqrt(tf.cast(d\_model, tf.float32))    embeddings = PositionalEncoding(vocab\_size, d\_model)(embeddings)    outputs = tf.keras.layers.Dropout(rate=dropout)(embeddings)    # 인코더를 num\_layers개 쌓기    for i in range(num\_layers):      outputs = encoder\_layer(dff=dff, d\_model=d\_model, num\_heads=num\_heads,          dropout=dropout, name="encoder\_layer\_{}".format(i),      )([outputs, padding\_mask])    return tf.keras.Model(        inputs=[inputs, padding\_mask], outputs=outputs, name=name)  def create\_look\_ahead\_mask(x):    seq\_len = tf.shape(x)[1]    look\_ahead\_mask = 1 - tf.linalg.band\_part(tf.ones((seq\_len, seq\_len)), -1, 0)    padding\_mask = create\_padding\_mask(x) # 패딩 마스크도 포함    return tf.maximum(look\_ahead\_mask, padding\_mask)  def decoder\_layer(dff, d\_model, num\_heads, dropout, name="decoder\_layer"):    inputs = tf.keras.Input(shape=(None, d\_model), name="inputs")    enc\_outputs = tf.keras.Input(shape=(None, d\_model), name="encoder\_outputs")    # 룩어헤드 마스크(첫번째 서브층)    look\_ahead\_mask = tf.keras.Input(        shape=(1, None, None), name="look\_ahead\_mask")    # 패딩 마스크(두번째 서브층)    padding\_mask = tf.keras.Input(shape=(1, 1, None), name='padding\_mask')    # 멀티-헤드 어텐션 (첫번째 서브층 / 마스크드 셀프 어텐션)    attention1 = MultiHeadAttention(        d\_model, num\_heads, name="attention\_1")(inputs={            'query': inputs, 'key': inputs, 'value': inputs, # Q = K = V            'mask': look\_ahead\_mask # 룩어헤드 마스크        })    # 잔차 연결과 층 정규화    attention1 = tf.keras.layers.LayerNormalization(        epsilon=1e-6)(attention1 + inputs)    # 멀티-헤드 어텐션 (두번째 서브층 / 디코더-인코더 어텐션)    attention2 = MultiHeadAttention(        d\_model, num\_heads, name="attention\_2")(inputs={            'query': attention1, 'key': enc\_outputs, 'value': enc\_outputs, # Q != K = V            'mask': padding\_mask # 패딩 마스크        })    # 드롭아웃 + 잔차 연결과 층 정규화    attention2 = tf.keras.layers.Dropout(rate=dropout)(attention2)    attention2 = tf.keras.layers.LayerNormalization(        epsilon=1e-6)(attention2 + attention1)    # 포지션 와이즈 피드 포워드 신경망 (세번째 서브층)    outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention2)    outputs = tf.keras.layers.Dense(units=d\_model)(outputs)    # 드롭아웃 + 잔차 연결과 층 정규화    outputs = tf.keras.layers.Dropout(rate=dropout)(outputs)    outputs = tf.keras.layers.LayerNormalization(        epsilon=1e-6)(outputs + attention2)    return tf.keras.Model(        inputs=[inputs, enc\_outputs, look\_ahead\_mask, padding\_mask],        outputs=outputs,        name=name)  def decoder(vocab\_size, num\_layers, dff,              d\_model, num\_heads, dropout,              name='decoder'):    inputs = tf.keras.Input(shape=(None,), name='inputs')    enc\_outputs = tf.keras.Input(shape=(None, d\_model), name='encoder\_outputs')    # 디코더는 룩어헤드 마스크(첫번째 서브층)와 패딩 마스크(두번째 서브층) 둘 다 사용.    look\_ahead\_mask = tf.keras.Input(        shape=(1, None, None), name='look\_ahead\_mask')    padding\_mask = tf.keras.Input(shape=(1, 1, None), name='padding\_mask')    # 포지셔널 인코딩 + 드롭아웃    embeddings = tf.keras.layers.Embedding(vocab\_size, d\_model)(inputs)    embeddings \*= tf.math.sqrt(tf.cast(d\_model, tf.float32))    embeddings = PositionalEncoding(vocab\_size, d\_model)(embeddings)    outputs = tf.keras.layers.Dropout(rate=dropout)(embeddings)    # 디코더를 num\_layers개 쌓기    for i in range(num\_layers):      outputs = decoder\_layer(dff=dff, d\_model=d\_model, num\_heads=num\_heads,          dropout=dropout, name='decoder\_layer\_{}'.format(i),      )(inputs=[outputs, enc\_outputs, look\_ahead\_mask, padding\_mask])    return tf.keras.Model(        inputs=[inputs, enc\_outputs, look\_ahead\_mask, padding\_mask],        outputs=outputs,        name=name)  def transformer(vocab\_size, num\_layers, dff,                  d\_model, num\_heads, dropout,                  name="transformer"):    # 인코더의 입력    inputs = tf.keras.Input(shape=(None,), name="inputs")    # 디코더의 입력    dec\_inputs = tf.keras.Input(shape=(None,), name="dec\_inputs")    # 인코더의 패딩 마스크    enc\_padding\_mask = tf.keras.layers.Lambda(        create\_padding\_mask, output\_shape=(1, 1, None),        name='enc\_padding\_mask')(inputs)    # 디코더의 룩어헤드 마스크(첫번째 서브층)    look\_ahead\_mask = tf.keras.layers.Lambda(        create\_look\_ahead\_mask, output\_shape=(1, None, None),        name='look\_ahead\_mask')(dec\_inputs)    # 디코더의 패딩 마스크(두번째 서브층)    dec\_padding\_mask = tf.keras.layers.Lambda(        create\_padding\_mask, output\_shape=(1, 1, None),        name='dec\_padding\_mask')(inputs)    # 인코더의 출력은 enc\_outputs. 디코더로 전달된다.    enc\_outputs = encoder(vocab\_size=vocab\_size, num\_layers=num\_layers, dff=dff,        d\_model=d\_model, num\_heads=num\_heads, dropout=dropout,    )(inputs=[inputs, enc\_padding\_mask]) # 인코더의 입력은 입력 문장과 패딩 마스크    # 디코더의 출력은 dec\_outputs. 출력층으로 전달된다.    dec\_outputs = decoder(vocab\_size=vocab\_size, num\_layers=num\_layers, dff=dff,        d\_model=d\_model, num\_heads=num\_heads, dropout=dropout,    )(inputs=[dec\_inputs, enc\_outputs, look\_ahead\_mask, dec\_padding\_mask])    # 다음 단어 예측을 위한 출력층    outputs = tf.keras.layers.Dense(units=vocab\_size, name="outputs")(dec\_outputs)    return tf.keras.Model(inputs=[inputs, dec\_inputs], outputs=outputs, name=name)  small\_transformer = transformer(      vocab\_size = 9000,      num\_layers = 4,      dff = 512,      d\_model = 128,      num\_heads = 4,      dropout = 0.3,      name="small\_transformer")  tf.keras.utils.plot\_model(      small\_transformer, to\_file='small\_transformer.png', show\_shapes=True)  def loss\_function(y\_true, y\_pred):    y\_true = tf.reshape(y\_true, shape=(-1, MAX\_LENGTH - 1))    loss = tf.keras.losses.SparseCategoricalCrossentropy(        from\_logits=True, reduction='none')(y\_true, y\_pred)    mask = tf.cast(tf.not\_equal(y\_true, 0), tf.float32)    loss = tf.multiply(loss, mask)    return tf.reduce\_mean(loss)  class CustomSchedule(tf.keras.optimizers.schedules.LearningRateSchedule):    def \_\_init\_\_(self, d\_model, warmup\_steps=4000):      super(CustomSchedule, self).\_\_init\_\_()      self.d\_model = d\_model      self.d\_model = tf.cast(self.d\_model, tf.float32)      self.warmup\_steps = warmup\_steps    def \_\_call\_\_(self, step):      arg1 = tf.math.rsqrt(step)      arg2 = step \* (self.warmup\_steps\*\*-1.5)      return tf.math.rsqrt(self.d\_model) \* tf.math.minimum(arg1, arg2)  sample\_learning\_rate = CustomSchedule(d\_model=128)  plt.plot(sample\_learning\_rate(tf.range(200000, dtype=tf.float32)))  plt.ylabel("Learning Rate")  plt.xlabel("Train Step")  import pandas as pd  import numpy as np  import matplotlib.pyplot as plt  import re  import urllib.request  import time  import tensorflow\_datasets as tfds  import tensorflow as tf  urllib.request.urlretrieve("https://raw.githubusercontent.com/  songys/Chatbot\_data/master/ChatbotData.csv", filename="ChatBotData.csv")  train\_data = pd.read\_csv(‘ChatbotData.csv’)  questions = []  for sentence in train\_data['Q']:      # 구두점에 대해서 띄어쓰기      # ex) 12시 땡! -> 12시 땡 !      sentence = re.sub(r"([?.!,])", r" \1 ", sentence)      sentence = sentence.strip()      questions.append(sentence)  answers = []  for sentence in train\_data['A']:      # 구두점에 대해서 띄어쓰기      # ex) 12시 땡! -> 12시 땡 !      sentence = re.sub(r"([?.!,])", r" \1 ", sentence)      sentence = sentence.strip()      answers.append(sentence)  # 서브워드텍스트인코더를 사용하여 질문, 답변 데이터로부터 단어 집합(Vocabulary) 생성  tokenizer = tfds.deprecated.text.SubwordTextEncoder.build\_from\_corpus(      questions + answers, target\_vocab\_size=2\*\*13)  # 시작 토큰과 종료 토큰에 대한 정수 부여.  START\_TOKEN, END\_TOKEN = [tokenizer.vocab\_size], [tokenizer.vocab\_size + 1]  # 시작 토큰과 종료 토큰을 고려하여 단어 집합의 크기를 + 2  VOCAB\_SIZE = tokenizer.vocab\_size + 2  # 서브워드텍스트인코더 토크나이저의 .encode()와 .decode() 테스트해보기  # 임의의 입력 문장을 sample\_string에 저장  sample\_string = questions[20]  # encode() : 텍스트 시퀀스 --> 정수 시퀀스  tokenized\_string = tokenizer.encode(sample\_string)  print ('정수 인코딩 후의 문장 {}'.format(tokenized\_string))  # decode() : 정수 시퀀스 --> 텍스트 시퀀스  original\_string = tokenizer.decode(tokenized\_string)  print ('기존 문장: {}'.format(original\_string))  # 각 정수는 각 단어와 어떻게 mapping되는지 병렬로 출력  # 서브워드텍스트인코더는 의미있는 단위의 서브워드로 토크나이징한다. 띄어쓰기 단위 X 형태소 분석 단위 X  for ts in tokenized\_string:    print ('{} ----> {}'.format(ts, tokenizer.decode([ts])))  # 최대 길이를 40으로 정의  MAX\_LENGTH = 40  # 토큰화 / 정수 인코딩 / 시작 토큰과 종료 토큰 추가 / 패딩  def tokenize\_and\_filter(inputs, outputs):    tokenized\_inputs, tokenized\_outputs = [], []    for (sentence1, sentence2) in zip(inputs, outputs):      # encode(토큰화 + 정수 인코딩), 시작 토큰과 종료 토큰 추가      sentence1 = START\_TOKEN + tokenizer.encode(sentence1) + END\_TOKEN      sentence2 = START\_TOKEN + tokenizer.encode(sentence2) + END\_TOKEN      tokenized\_inputs.append(sentence1)      tokenized\_outputs.append(sentence2)    # 패딩    tokenized\_inputs = tf.keras.preprocessing.sequence.pad\_sequences(        tokenized\_inputs, maxlen=MAX\_LENGTH, padding='post')    tokenized\_outputs = tf.keras.preprocessing.sequence.pad\_sequences(        tokenized\_outputs, maxlen=MAX\_LENGTH, padding='post')    return tokenized\_inputs, tokenized\_outputs  questions, answers = tokenize\_and\_filter(questions, answers)  # 텐서플로우 dataset을 이용하여 셔플(shuffle)을 수행하되, 배치 크기로 데이터를 묶는다.  # 또한 이 과정에서 교사 강요(teacher forcing)을 사용하기 위해서 디코더의 입력과 실제값 시퀀스를 구성한다.  BATCH\_SIZE = 64  BUFFER\_SIZE = 20000  # 디코더의 실제값 시퀀스에서는 시작 토큰을 제거해야 한다.  dataset = tf.data.Dataset.from\_tensor\_slices((      {          'inputs': questions,          'dec\_inputs': answers[:, :-1] # 디코더의 입력. 마지막 패딩 토큰이 제거된다.      },      {          'outputs': answers[:, 1:]  # 맨 처음 토큰이 제거된다. 다시 말해 시작 토큰이 제거된다.      },  ))  dataset = dataset.cache()  dataset = dataset.shuffle(BUFFER\_SIZE)  dataset = dataset.batch(BATCH\_SIZE)  dataset = dataset.prefetch(tf.data.experimental.AUTOTUNE)  tf.keras.backend.clear\_session()  # 하이퍼파라미터  D\_MODEL = 256  NUM\_LAYERS = 2  NUM\_HEADS = 8  DFF = 512  DROPOUT = 0.1  model = transformer(      vocab\_size=VOCAB\_SIZE,      num\_layers=NUM\_LAYERS,      dff=DFF,      d\_model=D\_MODEL,      num\_heads=NUM\_HEADS,      dropout=DROPOUT)  learning\_rate = CustomSchedule(D\_MODEL)  optimizer = tf.keras.optimizers.Adam(      learning\_rate, beta\_1=0.9, beta\_2=0.98, epsilon=1e-9)  def accuracy(y\_true, y\_pred):    # 레이블의 크기는 (batch\_size, MAX\_LENGTH - 1)    y\_true = tf.reshape(y\_true, shape=(-1, MAX\_LENGTH - 1))    return tf.keras.metrics.sparse\_categorical\_accuracy(y\_true, y\_pred)  model.compile(optimizer=optimizer, loss=loss\_function, metrics=[accuracy])  EPOCHS = 50  model.fit(dataset, epochs=EPOCHS)  def preprocess\_sentence(sentence):    # 단어와 구두점 사이에 공백 추가.    # ex) 12시 땡! -> 12시 땡 !    sentence = re.sub(r"([?.!,])", r" \1 ", sentence)    sentence = sentence.strip()  return sentence  def evaluate(sentence):    # 입력 문장에 대한 전처리    sentence = preprocess\_sentence(sentence)    # 입력 문장에 시작 토큰과 종료 토큰을 추가    sentence = tf.expand\_dims(        START\_TOKEN + tokenizer.encode(sentence) + END\_TOKEN, axis=0)    output = tf.expand\_dims(START\_TOKEN, 0)    # 디코더의 예측 시작    for i in range(MAX\_LENGTH):      predictions = model(inputs=[sentence, output], training=False)      # 현재 시점의 예측 단어를 받아온다.      predictions = predictions[:, -1:, :]      predicted\_id = tf.cast(tf.argmax(predictions, axis=-1), tf.int32)      # 만약 현재 시점의 예측 단어가 종료 토큰이라면 예측을 중단      if tf.equal(predicted\_id, END\_TOKEN[0]):        break      # 현재 시점의 예측 단어를 output(출력)에 연결한다.      # output은 for문의 다음 루프에서 디코더의 입력이 된다.      output = tf.concat([output, predicted\_id], axis=-1)    # 단어 예측이 모두 끝났다면 output을 리턴.    return tf.squeeze(output, axis=0)  def predict(sentence):    prediction = evaluate(sentence)    # prediction == 디코더가 리턴한 챗봇의 대답에 해당하는 정수 시퀀스    # tokenizer.decode()를 통해 정수 시퀀스를 문자열로 디코딩.    predicted\_sentence = tokenizer.decode(        [i for i in prediction if i < tokenizer.vocab\_size])    print('Input: {}'.format(sentence))    print('Output: {}'.format(predicted\_sentence))    return predicted\_sentence  output = predict("영화 볼래?")  output = predict("고민이 있어")  output = predict("너무 화가나")  output = predict("카페갈래?")  output = predict("게임하고싶당")  output = predict("게임하자") |

# testing result

|  |
| --- |
| Epoch 1/50  185/185 [==============================] - 16s 55ms/step - loss: 1.4533 - accuracy: 0.0278  Epoch 2/50  185/185 [==============================] - 10s 55ms/step - loss: 1.1824 - accuracy: 0.0494  Epoch 3/50  185/185 [==============================] - 10s 55ms/step - loss: 1.0042 - accuracy: 0.0506  Epoch 4/50  185/185 [==============================] - 10s 54ms/step - loss: 0.9288 - accuracy: 0.0543  Epoch 5/50  185/185 [==============================] - 10s 55ms/step - loss: 0.8706 - accuracy: 0.0576  Epoch 6/50  185/185 [==============================] - 10s 55ms/step - loss: 0.8126 - accuracy: 0.0615  Epoch 7/50  185/185 [==============================] - 10s 55ms/step - loss: 0.7487 - accuracy: 0.0671  Epoch 8/50  185/185 [==============================] - 10s 56ms/step - loss: 0.6751 - accuracy: 0.0751  Epoch 9/50  185/185 [==============================] - 10s 57ms/step - loss: 0.5956 - accuracy: 0.0838  Epoch 10/50  185/185 [==============================] - 10s 56ms/step - loss: 0.5124 - accuracy: 0.0931  Epoch 11/50  185/185 [==============================] - 10s 55ms/step - loss: 0.4281 - accuracy: 0.1038  Epoch 12/50  185/185 [==============================] - 10s 56ms/step - loss: 0.3472 - accuracy: 0.1149  Epoch 13/50  185/185 [==============================] - 11s 57ms/step - loss: 0.0060 - accuracy: 0.1736 Epoch 49/50 185/185 [==============================] - 10s 56ms/step - loss: 0.0056 - accuracy: 0.1738 Epoch 50/50 185/185 [==============================] - 10s 56ms/step - loss: 0.0055 - accuracy: 0.1737  Input: 영화 볼래? Output: 최신 영화가 좋을 것 같아요 .  Input: 너무 화가나 Output: 그럴수록 당신이 힘들 거예요 .  Input: 카페갈래? Output: 쪽지를 남겨서 마음을 전해보세요 .  Input: 게임하고싶당 Output: 그럴 수 있어요 .  Input: 게임하자 Output: 게임하세요 ! |