Chpater6. 챗봇 만들기

일반적으로 공개된 챗봇을 위한 한글 데이터는 거의 없다. 데이터로는 'Chatbot_data_for_Korean v1.0'(https://github.com/songys/Chatbot_data)를 사용하였다. 규칙 기반으로 제작할 수 있고 머신러닝을 활용한 유사도 기반, 규칙과 머신러닝을 섞은 하이브리드 기반, 특정 시나리오에서 동작이 가능해지는 시나리오 기반 등 정의하는 사람에 따라 챗봇 제작 방법은 다양하다. 이 책에서는 Sequence-to-Sequence, 트랜스포머 모델을 이용하여 제작하였다.

Ⅰ. 시퀀스 투 시퀀스 모델

1. Preprocess.py

데이터를 불러오고 가공한다.

```
FILTERS="([~.,!?₩"':;)(])"
PAD="<PAD>" #의미없음
STD="<SOS>" #시작 토큰
END="<END>" #종료 토큰
UNK="<UNK>" #사전에 없는 단어

PAD_INDEX=0
STD_INDEX=1
END_INDEX=2
UNK_INDEX=3

MARKER=[PAD, STD, END, UNK]
CHANGE_FILTER=re.compile(FILTERS)
MAX_SEQUENCE=25
```

```
PATH='/content/drive/MyDrive/Kaggle/챗봇/ChatbotData .csv'
VOCAB_PATH='/content/drive/MyDrive/Kaggle/챗봇/vocabulary_list.csv'
```

```
def load_data(path): #데이터셋을 question과 answer로 나누기
  data_df=pd.read_csv(path, header=0)
  question, answer=list(data_df['Q']), list(data_df['A'])
  return question, answer
```

```
def data_tokenizer(data): #데이터 전처리 후 단어 리스트 만들기
  words=[]
  for sentence in data:
    sentence=re.sub(CHANGE_FILTER, "", sentence)
    for word in sentence.split():
      words.append(word)
  return [word for word in words if word]
```

```
def prepro_like_morphlized(data): #형태소로 분리
  morph_analyzer=Okt()
  result_data=list()
  for seq in tqdm(data):
    morphlized_seq=" ".join(morph_analyzer.morphs(seq.replace(' ','')))
    result_data.append(morphlized_seq)
  return result_data
```

```
def load_vocabulary(path, vocab_path, tokenize_as_morph=False): #단어 사전 만들기
  vocabulary_list=[]
  #단어 사전 파일이 없으면 prepro_like_morphlized를 이용해 토크나이징해서 단어 리스트를 만든다.
  if not os.path.exists(vocab_path):
    if (os.path.exists(path)):
      data_df=pd.read_csv(path, encoding='utf-8')
      question, answer=list(data_df['Q']), list(data_df['A'])
      if tokenize_as_morph:
        question=prepro_like_morphlized(question)
        answer=prepro_like_morphlized(answer)
      data=[]
      data.extend(question)
      data.extend(answer)
      words=data_tokenizer(data)
      words=list(set(words))
      words[:0]=MARKER #사전에 정의한 특정 토큰들을 단어 리스트 앞에 추가
    with open(vocab_path, 'w', encoding='utf-8') as vocabulary_file:
      for word in words:
        vocabulary_file.write(word+'₩n')
  with open(vocab_path, 'r', encoding='utf-8') as vocabulary_file:
    for line in vocabulary_file:
      vocabulary_list.append(line.strip())
  word2idx, idx2word=make_vocabulary(vocabulary_list)
  return word2idx, idx2word, len(word2idx)
  #word2idx : 단어에 대한 인덱스
  #idx2word : 인덱스에 대한 단어를 가진 딕셔너리 데이터
```

```
def make_vocabulary(vocabulary_list):
  #리스트를 키가 단어이고 값이 인덱스인 딕셔너리를 만든다
  word2idx={word:idx for idx, word in enumerate(vocabulary_list)}
  #리스트를 키가 인덱스이고 값이 단어인 딕셔너리를 만든다
  idx2word={idx:word for idx, word in enumerate(vocabulary_list)}
  return word2idx, idx2word
word2idx, idx2word, vocab_size=load_vocabulary(PATH, VOCAB_PATH)
```

```
def enc_processing(value, dictionary, tokenize_as_morph=False):
  #value : 전처리할 데이터(띄어쓰기를 기준으로)
  #dictionary : 단어 사전
  sequences_input_index=[]
  sequences_length=[]
  if tokenize_as_morph:
    value=prepro_like_morphlized(value)
  for sequence in value:
    sequence=re.sub(CHANGE_FILTER, """", sequence) #특수문자 모두 제거
    sequence_index=[]
    for word in sequence.split():
      if dictionary.get(word) is not None:
        sequence_index.extend([dictionary[word]]) #단어 사전을 이용해 단어 인덱스로 바꾼다
      else:
        sequence_index.extend([dictionary[UNK]]) #단어 사전에 포함되어 있지 않으면 UNK
    if len(sequence_index)>MAX_SEQUENCE:
      sequence_index=sequence_index[:MAX_SEQUENCE]
    sequences_length.append(len(sequence_index))
    sequence_index+=(MAX_SEQUENCE-len(sequence_index))*[dictionary[PAD]]
    sequences_input_index.append(sequence_index)
  return np.asarray(sequences_input_index), sequences_length #전처리한 데이터, 각 문장의 실제 길이
```

```
#디코더의 입력값 함수
#각 문자의 처음에 시작 토큰을 넣는다.
#데이터와 단어 사전을 인자로 받고 전처리한 데이터와 문장의 실제 길이의 리스트 리턴
def dec_output_processing(value, dictionary, tokenize_as_morph=False):
  sequences_output_index=[]
  sequences_length=[]
  if tokenize_as_morph:
    value=prepro_like_morphlized(value)
  for sequence in value:
    sequence=re.sub(CHANGE_FILTER, """", sequence)
    sequence_index=[]
    sequence_index=[dictionary[STD]]+[dictionary[word] if word in dictionary
                                     else dictionary[UNK] for word in sequence.split()]
    if len(sequence_index)>MAX_SEQUENCE:
      sequence_index=sequence_index[:MAX_SEQUENCE]
    sequences_length.append(len(sequence_index))
    sequence_index+=(MAX_SEQUENCE-len(sequence_index))*[dictionary[PAD]]
    sequences_output_index.append(sequence_index)
  return np.array(sequences_output_index), sequences_length
```

```
#데이터와 단어 사전을 인자로 받고 전처리한 데이터와 각 데이터 문장의 실제 길이의 리스트를 리턴
#문장이 시작하는 부분에 시작 토큰을 넣지 않고 마지막에 종료 토큰을 넣는다.
def dec_target_processing(value, dictionary, tokenize_as_morph=False):
  sequences_target_index=[]
  if tokenize_as_morph:
    value=prepro_like_morphlized(value)
  for sequence in value:
    sequence=re.sub(CHANGE_FILTER, """", sequence)
    sequence_index=[dictionary[word] if word in dictionary else dictionary[UNK] for word in sequence.split()]
    if len(sequence_index)>=MAX_SEQUENCE:
      sequence_index=sequence_index[:MAX_SEQUENCE-1]+[dictionary[END]]
    else:
      sequence_index+=[dictionary[END]]
    sequence_index+=(MAX_SEQUENCE-len(sequence_index))*[dictionary[PAD]]
    sequences_target_index.append(sequence_index)
  return np.asarray(sequences_target_index)
```

2. Preprocess.ipynb

학습 데이터 준비

```
inputs, outputs=load_data(PATH)
char2idx, idx2char, vocab_size = load_vocabulary(PATH, VOCAB_PATH, tokenize_as_morph=False)
#False면 띄어쓰기 단위, True면 형태소 단위
```

```
index_inputs, input_seq_len = enc_processing(inputs, char2idx, tokenize_as_morph=False)
index_outputs, output_seq_len = dec_output_processing(outputs, char2idx, tokenize_as_morph=False)
index_targets = dec_target_processing(outputs, char2idx, tokenize_as_morph=False)
```

```
data_configs = {}
data_configs['char2idx'] = char2idx
data_configs['idx2char'] = idx2char
data_configs['vocab_size'] = vocab_size
data_configs['pad_symbol'] = PAD
data_configs['std_symbol'] = STD
data_configs['end_symbol'] = END
data_configs['unk_symbol'] = UNK
```

```
DATA_IN_PATH = '/content/drive/MyDrive/Kaggle/챗봇/'
TRAIN_INPUTS = 'train_inputs.npy'
TRAIN_OUTPUTS = 'train_outputs.npy'
TRAIN_TARGETS = 'train_targets.npy'
DATA_CONFIGS = 'data_configs.json'

np.save(open(DATA_IN_PATH + TRAIN_INPUTS, 'wb'), index_inputs)
np.save(open(DATA_IN_PATH + TRAIN_OUTPUTS , 'wb'), index_outputs)
np.save(open(DATA_IN_PATH + TRAIN_TARGETS , 'wb'), index_targets)

json.dump(data_configs, open(DATA_IN_PATH + DATA_CONFIGS, 'w'))
```

3. seq2seq.ipynb

```
SEED_NUM = 1234
tf.random.set_seed(SEED_NUM)
```

```
index_inputs = np.load(open(DATA_IN_PATH + TRAIN_INPUTS, 'rb'))
index_outputs = np.load(open(DATA_IN_PATH + TRAIN_OUTPUTS , 'rb'))
index_targets = np.load(open(DATA_IN_PATH + TRAIN_TARGETS , 'rb'))
prepro_configs = json.load(open(DATA_IN_PATH + DATA_CONFIGS, 'r'))
```

```
MODEL_NAME = 'seq2seq_kor'
BATCH_SIZE = 2
MAX_SEQUENCE = 25
EPOCH = 30
UNITS = 1024
EMBEDDING_DIM = 256
VALIDATION_SPLIT = 0.1

char2idx = prepro_configs['char2idx']
idx2char = prepro_configs['idx2char']
std_index = prepro_configs['std_symbol']
end_index = prepro_configs['end_symbol']
vocab_size = prepro_configs['vocab_size']
```

```
#신경망으로 GRU사용함
class Encoder(tf.keras.layers.Layer):
    def __init__(self, vocab_size, embedding_dim, enc_units, batch_sz):
        super(Encoder, self).__init__()
        self.batch_sz = batch_sz
        self.enc_units = enc_units
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim

        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)
        self.gru = tf.keras.layers.GRU(self.enc_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')

    def call(self, x, hidden):
        x = self.embedding(x)
        output, state = self.gru(x, initial_state = hidden)
        return output, state

    def initialize_hidden_state(self, inp):
        return tf.zeros((tf.shape(inp)[0], self.enc_units))
```

```
class BahdanauAttention(tf.keras.layers.Layer):
    def __init__(self, units):
        super(BahdanauAttention, self).__init__()
        self.W1 = tf.keras.layers.Dense(units)
        self.W2 = tf.keras.layers.Dense(units)
        self.V = tf.keras.layers.Dense(1)

    def call(self, query, values):
      #query : 은닉층의 상태 값
    #values : 인코더 재귀 순환망의 결괏값
        hidden_with_time_axis = tf.expand_dims(query, 1)

        score = self.V(tf.nn.tanh(
            self.W1(values) + self.W2(hidden_with_time_axis)))

        attention_weights = tf.nn.softmax(score, axis=1)
        #attention_weights : 어텐션 가중치
        context_vector = attention_weights * values
        context_vector = tf.reduce_sum(context_vector, axis=1)

        return context_vector, attention_weights
```

```
class Decoder(tf.keras.layers.Layer):
    def __init__(self, vocab_size, embedding_dim, dec_units, batch_sz):
        super(Decoder, self).__init__()

        self.batch_sz = batch_sz
        self.dec_units = dec_units
        self.vocab_size = vocab_size
        self.embedding_dim = embedding_dim

        self.embedding = tf.keras.layers.Embedding(self.vocab_size, self.embedding_dim)
        self.gru = tf.keras.layers.GRU(self.dec_units,
                                       return_sequences=True,
                                       return_state=True,
                                       recurrent_initializer='glorot_uniform')
        self.fc = tf.keras.layers.Dense(self.vocab_size)

        self.attention = BahdanauAttention(self.dec_units)

    def call(self, x, hidden, enc_output):
        context_vector, attention_weights = self.attention(hidden, enc_output)

        x = self.embedding(x)

        x = tf.concat([tf.expand_dims(context_vector, 1), x], axis=-1)

        output, state = self.gru(x)
        output = tf.reshape(output, (-1, output.shape[2]))

        x = self.fc(output)

        return x, state, attention_weights
```

```
#손실함수, 정확도 측정 함수
optimizer = tf.keras.optimizers.Adam()

loss_object = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True, reduction='none')

train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')

def loss(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)
    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask
    return tf.reduce_mean(loss_)

def accuracy(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    mask = tf.expand_dims(tf.cast(mask, dtype=pred.dtype), axis=-1)
    pred *= mask
    acc = train_accuracy(real, pred)

    return tf.reduce_mean(acc)
```

```
#seq2seq 메인 클래스 : 각각 분리돼 있는 각 클래스를 이어줌
class seq2seq(tf.keras.Model):
    def __init__(self, vocab_size, embedding_dim, enc_units, dec_units, batch_sz, end_token_idx=2):
        super(seq2seq, self).__init__()
        self.end_token_idx = end_token_idx
        self.encoder = Encoder(vocab_size, embedding_dim, enc_units, batch_sz)
        self.decoder = Decoder(vocab_size, embedding_dim, dec_units, batch_sz)

    def call(self, x):
        inp, tar = x

        enc_hidden = self.encoder.initialize_hidden_state(inp)
        enc_output, enc_hidden = self.encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        predict_tokens = list()
        for t in range(0, tar.shape[1]):
            dec_input = tf.dtypes.cast(tf.expand_dims(tar[:, t], 1), tf.float32)
            predictions, dec_hidden, _ = self.decoder(dec_input, dec_hidden, enc_output)
            predict_tokens.append(tf.dtypes.cast(predictions, tf.float32))
        return tf.stack(predict_tokens, axis=1)
```

```
    def inference(self, x):
        inp  = x

        enc_hidden = self.encoder.initialize_hidden_state(inp)
        enc_output, enc_hidden = self.encoder(inp, enc_hidden)

        dec_hidden = enc_hidden

        dec_input = tf.expand_dims([char2idx[std_index]], 1)

        predict_tokens = list()
        for t in range(0, MAX_SEQUENCE):
            predictions, dec_hidden, _ = self.decoder(dec_input, dec_hidden, enc_output)
            predict_token = tf.argmax(predictions[0])

            if predict_token == self.end_token_idx:
                break

            predict_tokens.append(predict_token)
            dec_input = tf.dtypes.cast(tf.expand_dims([predict_token], 0), tf.float32)

        return tf.stack(predict_tokens, axis=0).numpy()
```

```
model = seq2seq(vocab_size, EMBEDDING_DIM, UNITS, UNITS, BATCH_SIZE, char2idx[end_index])
model.compile(loss=loss, optimizer=tf.keras.optimizers.Adam(1e-3), metrics=[accuracy])
```

모델 학습

```
PATH = DATA_IN_PATH + MODEL_NAME
if not(os.path.isdir(PATH)):
        os.makedirs(os.path.join(PATH))

checkpoint_path = DATA_IN_PATH + MODEL_NAME + '/weights.h5'

cp_callback = ModelCheckpoint(
    checkpoint_path, monitor='val_accuracy', verbose=1, save_best_only=True, save_weights_only=True)

earlystop_callback = EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=10)

history = model.fit([index_inputs, index_outputs], index_targets,
                    batch_size=BATCH_SIZE, epochs=EPOCH,
                    validation_split=VALIDATION_SPLIT, callbacks=[earlystop_callback, cp_callback])
```

```
Epoch 1/30
5320/5320 [==============================] - 1057s 195ms/step - loss: 1.0053 - accuracy: 0.8886 - val_loss: 1.

Epoch 00001: val_accuracy improved from -inf to 0.89256, saving model to /content/drive/MyDrive/Kaggle/챗봇/se
Epoch 2/30
5320/5320 [==============================] - 1015s 191ms/step - loss: 1.0170 - accuracy: 0.8923 - val_loss: 1.

Epoch 00002: val_accuracy did not improve from 0.89256
Epoch 3/30
5320/5320 [==============================] - 1010s 190ms/step - loss: 1.0221 - accuracy: 0.8924 - val_loss: 1.

Epoch 00003: val_accuracy did not improve from 0.89256
Epoch 4/30
5320/5320 [==============================] - 1011s 190ms/step - loss: 0.9660 - accuracy: 0.8923 - val_loss: 1.

Epoch 00004: val_accuracy did not improve from 0.89256
Epoch 5/30
5320/5320 [==============================] - 1013s 190ms/step - loss: 0.8927 - accuracy: 0.8924 - val_loss: 1.

Epoch 00005: val_accuracy did not improve from 0.89256
Epoch 6/30
5320/5320 [==============================] - 1008s 190ms/step - loss: 0.7091 - accuracy: 0.8925 - val_loss: 1.
```

```
SAVE_FILE_NM="weights.h5"
model.load_weights(os.path.join(DATA_IN_PATH, MODEL_NAME, SAVE_FILM_NM))
```

```
query = "남자친구 승진 선물로 뭐가 좋을까?"

test_index_inputs, _ = enc_processing([query], char2idx)
predict_tokens = model.inference(test_index_inputs)
print(predict_tokens)

print(' '.join([idx2char[str(t)] for t in predict_tokens]))
```

학습 과정이 굉장히 오래 걸리고(대략 12시간 정도) 집안 와이파이 상황이 안좋아 Colab 에서 돌리는 것이 무리였다. 그래서 결과는 보지 못했다.

Ⅱ. 트랜스포머 모델

Preprocess.py와 Preprocess.ipynb는 위와 같은 파일을 이용하지만 한 가지 다른 점은 띄어쓰기가 아닌 형태소 단위로 토크나이즈하는 방식으로 전처리한다.

```
index_inputs, input_seq_len = enc_processing(inputs, char2idx, tokenize_as_morph=True)
index_outputs, output_seq_len = dec_output_processing(outputs, char2idx, tokenize_as_morph=True)
index_targets = dec_target_processing(outputs, char2idx, tokenize_as_morph=True)
```

1. 패딩 및 포워드 마스킹

```
def create_padding_mask(seq):
    seq = tf.cast(tf.math.equal(seq, 0), tf.float32)

    # add extra dimensions to add the padding
    # to the attention logits.
    return seq[:, tf.newaxis, tf.newaxis, :]  # (batch_size, 1, 1, seq_len)
```

```
def create_look_ahead_mask(size):
  #행렬의 아래쪽 삼각형 영역이 00이 되는 하삼각행렬을 만든다.
  #tf.ones : 모든 값이 1인 행렬을 만든다.
  #마스킹하고자 하는 영역에 1값을 빼줌.
    mask = 1 - tf.linalg.band_part(tf.ones((size, size)), -1, 0)
    return mask  # (seq_len, seq_len)
```

```
def create_masks(inp, tar):
    # Encoder padding mask
    enc_padding_mask = create_padding_mask(inp)

    # Used in the 2nd attention block in the decoder.
    # This padding mask is used to mask the encoder outputs.
    dec_padding_mask = create_padding_mask(inp)

    # Used in the 1st attention block in the decoder.
    # It is used to pad and mask future tokens in the input received by
    # the decoder.
    look_ahead_mask = create_look_ahead_mask(tf.shape(tar)[1])
    dec_target_padding_mask = create_padding_mask(tar)
    combined_mask = tf.maximum(dec_target_padding_mask, look_ahead_mask)

    return enc_padding_mask, combined_mask, dec_padding_mask
```

```
enc_padding_mask, look_ahead_mask, dec_padding_mask = create_masks(index_inputs, index_outputs)
```

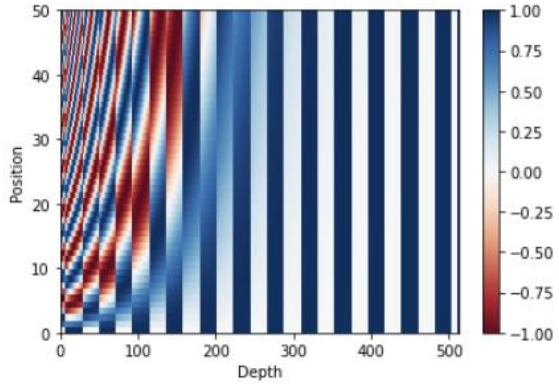
포지셔널 인코딩

```
#포지션 임베딩을 만들 행렬을 구성하기 위함.
def get_angles(pos, i, d_model):
    angle_rates = 1 / np.power(10000, (2 * i//2) / np.float32(d_model))
    return pos * angle_rates
```

```
def positional_encoding(position, d_model):
    angle_rads = get_angles(np.arange(position)[:, np.newaxis],
                            np.arange(d_model)[np.newaxis, :],
                            d_model)

    # apply sin to even indices in the array; 2i
    angle_rads[:, 0::2] = np.sin(angle_rads[:, 0::2])

    # apply cos to odd indices in the array; 2i+1
    angle_rads[:, 1::2] = np.cos(angle_rads[:, 1::2])

    pos_encoding = angle_rads[np.newaxis, ...]

    return tf.cast(pos_encoding, dtype=tf.float32)
```

```
pos_encoding = positional_encoding(50, 512)
print (pos_encoding.shape)

plt.pcolormesh(pos_encoding[0], cmap='RdBu')
plt.xlabel('Depth')
plt.xlim((0, 512))
plt.ylabel('Position')
plt.colorbar()
plt.show()
```

```
(1, 50, 512)
```



어텐션

```
#스케일 내적 어텐션 함수를 가지고 마스크 기법을 추가한 내적 함수
def scaled_dot_product_attention(q, k, v, mask):

    matmul_qk = tf.matmul(q, k, transpose_b=True)  # (..., seq_len_q, seq_len_k)

    # scale matmul_qk
    dk = tf.cast(tf.shape(k)[-1], tf.float32)
    scaled_attention_logits = matmul_qk / tf.math.sqrt(dk)

    # add the mask to the scaled tensor.
    if mask is not None:
        scaled_attention_logits += (mask * -1e9)

    # softmax is normalized on the last axis (seq_len_k) so that the scores
    # add up to 1.
    attention_weights = tf.nn.softmax(scaled_attention_logits, axis=-1)  # (..., seq_len_q, seq_len_k)

    output = tf.matmul(attention_weights, v)  # (..., seq_len_q, depth_v)

    return output, attention_weights
```

멀티헤드어텐션

```
class MultiHeadAttention(tf.keras.layers.Layer):
    def __init__(self, **kargs):
        super(MultiHeadAttention, self).__init__()
        self.num_heads = kargs['num_heads'] #어텐션 head 수
        self.d_model = kargs['d_model'] #key, query, value에 대한 차원을 정의하기 위한 파라미터

        assert self.d_model % self.num_heads == 0 #d_model의 차원 수는 헤드 개수만큼 나눠져야 하기 때문

        self.depth = self.d_model // self.num_heads
        #스케일 내적 연산 이전에 입력한 key, query, value에 대한 차원 수를 맞추기 위한 레이어
        self.wq = tf.keras.layers.Dense(kargs['d_model'])
        self.wk = tf.keras.layers.Dense(kargs['d_model'])
        self.wv = tf.keras.layers.Dense(kargs['d_model'])
        #셀프 어텐션 레이어를 출력하기 위한 레이어
        self.dense = tf.keras.layers.Dense(kargs['d_model'])
    #key, query, value에 대한 벡터를 헤드 수만큼 분리할 수 있게 하는 함수
    #[배치차원*시퀀스차원*피처차원]->[배치차원*헤드차원*시퀀스차원*피처차원]
    def split_heads(self, x, batch_size):
        #피처 차원을 헤드 수만큼 분리
        x = tf.reshape(x, (batch_size, -1, self.num_heads, self.depth))
        #transpose로 시퀀스, 헤드 차원만 바꾼다.
        return tf.transpose(x, perm=[0, 2, 1, 3])
```

```
    def call(self, v, k, q, mask):
        batch_size = tf.shape(q)[0]

        q = self.wq(q)  # (batch_size, seq_len, d_model)
        k = self.wk(k)  # (batch_size, seq_len, d_model)
        v = self.wv(v)  # (batch_size, seq_len, d_model)

        q = self.split_heads(q, batch_size)  # (batch_size, num_heads, seq_len_q, depth)
        k = self.split_heads(k, batch_size)  # (batch_size, num_heads, seq_len_k, depth)
        v = self.split_heads(v, batch_size)  # (batch_size, num_heads, seq_len_v, depth)

        # scaled_attention.shape == (batch_size, num_heads, seq_len_q, depth)
        # attention_weights.shape == (batch_size, num_heads, seq_len_q, seq_len_k)
        scaled_attention, attention_weights = scaled_dot_product_attention(
            q, k, v, mask)

        scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])  # (batch_size, seq_len_q, num_hea

        concat_attention = tf.reshape(scaled_attention,
                                      (batch_size, -1, self.d_model))  # (batch_size, seq_len_q, d_model)

        output = self.dense(concat_attention)  # (batch_size, seq_len_q, d_model)

        return output, attention_weights
```

포지션-와이즈 피드 포워드 네트워크

```
def point_wise_feed_forward_network(**kargs):
    return tf.keras.Sequential([
      tf.keras.layers.Dense(kargs['dff'], activation='relu'),  # (batch_size, seq_len, dff)
      tf.keras.layers.Dense(kargs['d_model'])  # (batch_size, seq_len, d_model)
    ])
```

인코더 레이어

```
class EncoderLayer(tf.keras.layers.Layer):
    def __init__(self, **kargs):
        super(EncoderLayer, self).__init__()
        #멀티헤드어텐션
        self.mha = MultiHeadAttention(**kargs)
        #포지션 와이즈 피드 포워드 네트워크
        self.ffn = point_wise_feed_forward_network(**kargs)
        #레이어 노멀라이제이션
        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        #드롭아웃
        self.dropout1 = tf.keras.layers.Dropout(kargs['rate'])
        self.dropout2 = tf.keras.layers.Dropout(kargs['rate'])

    def call(self, x, mask):
        attn_output, _ = self.mha(x, x, x, mask)  # (batch_size, input_seq_len, d_model)
        attn_output = self.dropout1(attn_output)
        out1 = self.layernorm1(x + attn_output)  # (batch_size, input_seq_len, d_model)

        ffn_output = self.ffn(out1)  # (batch_size, input_seq_len, d_model)
        ffn_output = self.dropout2(ffn_output)
        out2 = self.layernorm2(out1 + ffn_output)  # (batch_size, input_seq_len, d_model)

        return out2
```

인코더

```
#여러개의 인코더레이어를 쌓는다.
class Encoder(tf.keras.layers.Layer):
    def __init__(self, **kargs):
        super(Encoder, self).__init__()

        self.d_model = kargs['d_model']
        self.num_layers = kargs['num_layers']

        #input_vocab_size : 워드 임베딩의 사전 수
        #maximum_position_encoding : 포지션 인코더의 최대 시퀀스 길이
        self.embedding = tf.keras.layers.Embedding(kargs['input_vocab_size'], self.d_model)
        self.pos_encoding = positional_encoding(kargs['maximum_position_encoding'],
                                                self.d_model)

         #레이어 개수만큼 인코더 레이어를 생성해서 할당
        self.enc_layers = [EncoderLayer(**kargs)
                           for _ in range(self.num_layers)]
        #드롭아웃 레이어 생성
        self.dropout = tf.keras.layers.Dropout(kargs['rate'])

    def call(self, x, mask):

        seq_len = tf.shape(x)[1]

        # adding embedding and position encoding.
        x = self.embedding(x)  # (batch_size, input_seq_len, d_model)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x += self.pos_encoding[:, :seq_len, :]
        #포지션 임베딩은 행렬의 크기가 고정되어 있고 워드 임베딩은 입력 길이에 따라 가변적이다.
        #포지션 임베딩의 경우 워드 임베딩과 더할 경우 워드 임베딩의길이에 맞게 행렬 크기를 조절해야함.
        x = self.dropout(x)

        for i in range(self.num_layers):
            x = self.enc_layers[i](x, mask)

        return x  # (batch_size, input_seq_len, d_model)
```

디코더 레이어

```
#인코더에 하나의 레이어가 더 추가된 형태(2개의 어텐션+1개의 피드 포워드 레이어)
class DecoderLayer(tf.keras.layers.Layer):
    def __init__(self, **kargs):
        super(DecoderLayer, self).__init__()

        self.mha1 = MultiHeadAttention(**kargs)
        self.mha2 = MultiHeadAttention(**kargs)

        self.ffn = point_wise_feed_forward_network(**kargs)

        self.layernorm1 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm2 = tf.keras.layers.LayerNormalization(epsilon=1e-6)
        self.layernorm3 = tf.keras.layers.LayerNormalization(epsilon=1e-6)

        self.dropout1 = tf.keras.layers.Dropout(kargs['rate'])
        self.dropout2 = tf.keras.layers.Dropout(kargs['rate'])
        self.dropout3 = tf.keras.layers.Dropout(kargs['rate'])


    def call(self, x, enc_output, look_ahead_mask, padding_mask):
        # 디코더의 입력 사이의 관계를 계산하는 셀프 어텐션 구조
        # 마스크 어텐션 : 특정 단어 이후의 단어를 참고하지 않도록 마스크 기법 사용
        attn1, attn_weights_block1 = self.mha1(x, x, x, look_ahead_mask)  # (batch_size, target_seq_len, d_mode
        attn1 = self.dropout1(attn1)
        out1 = self.layernorm1(attn1 + x)

        attn2, attn_weights_block2 = self.mha2(
            enc_output, enc_output, out1, padding_mask)  # (batch_size, target_seq_len, d_model)
        attn2 = self.dropout2(attn2)
        out2 = self.layernorm2(attn2 + out1)  # (batch_size, target_seq_len, d_model)

        ffn_output = self.ffn(out2)  # (batch_size, target_seq_len, d_model)
        ffn_output = self.dropout3(ffn_output)
        out3 = self.layernorm3(ffn_output + out2)  # (batch_size, target_seq_len, d_model)

        return out3, attn_weights_block1, attn_weights_block2
```

```
#인코더 정보 벡터와 순방향 어텐션 마스크를 추가로 입력받는 것 외에 인코더와 같음.
class Decoder(tf.keras.layers.Layer):
    def __init__(self, **kargs):
        super(Decoder, self).__init__()

        self.d_model = kargs['d_model']
        self.num_layers = kargs['num_layers']

        self.embedding = tf.keras.layers.Embedding(kargs['target_vocab_size'], self.d_model)
        self.pos_encoding = positional_encoding(kargs['maximum_position_encoding'], self.d_model)

        self.dec_layers = [DecoderLayer(**kargs)
                           for _ in range(self.num_layers)]
        self.dropout = tf.keras.layers.Dropout(kargs['rate'])

    def call(self, x, enc_output, look_ahead_mask, padding_mask):
        seq_len = tf.shape(x)[1]
        attention_weights = {}

        x = self.embedding(x)  # (batch_size, target_seq_len, d_model)
        x *= tf.math.sqrt(tf.cast(self.d_model, tf.float32))
        x += self.pos_encoding[:, :seq_len, :]

        x = self.dropout(x)

        for i in range(self.num_layers):
            x, block1, block2 = self.dec_layers[i](x, enc_output, look_ahead_mask, padding_mask)

            attention_weights['decoder_layer{}_block1'.format(i+1)] = block1
            attention_weights['decoder_layer{}_block2'.format(i+1)] = block2

        # x.shape == (batch_size, target_seq_len, d_model)
        return x, attention_weights
```

트랜스포머 모델

```
class Transformer(tf.keras.Model):
    def __init__(self, **kargs):
        super(Transformer, self).__init__(name=kargs['model_name'])
        self.end_token_idx = kargs['end_token_idx']

        self.encoder = Encoder(**kargs) #인코더
        self.decoder = Decoder(**kargs) #디코더

        self.final_layer = tf.keras.layers.Dense(kargs['target_vocab_size'])
    #디코더에 입력할 시퀀스가 주어짐.
    def call(self, x):
      #inp : 인코더에 들어갈 값
      #tar : 디코더에 들어갈 값
        inp, tar = x

        enc_padding_mask, look_ahead_mask, dec_padding_mask = create_masks(inp, tar)
        enc_output = self.encoder(inp, enc_padding_mask)  # (batch_size, inp_seq_len, d_model)

        # dec_output.shape == (batch_size, tar_seq_len, d_model)
        dec_output, _ = self.decoder(
            tar, enc_output, look_ahead_mask, dec_padding_mask)

        final_output = self.final_layer(dec_output)  # (batch_size, tar_seq_len, target_vocab_size)

        return final_output
```

```
    #디코더에 입력할 시퀀스를 매번 생성해야함.
     def inference(self, x):
         inp = x
         tar = tf.expand_dims([STD_INDEX], 0)

         enc_padding_mask, look_ahead_mask, dec_padding_mask = create_masks(inp, tar)
         enc_output = self.encoder(inp, enc_padding_mask)

         predict_tokens = list()
         for t in range(0, MAX_SEQUENCE):
             dec_output, _ = self.decoder(tar, enc_output, look_ahead_mask, dec_padding_mask)
             final_output = self.final_layer(dec_output)
             outputs = tf.argmax(final_output, -1).numpy()
             pred_token = outputs[0][-1]
             if pred_token == self.end_token_idx:
                 break
             predict_tokens.append(pred_token)
             tar = tf.expand_dims([STD_INDEX] + predict_tokens, 0)
             _, look_ahead_mask, dec_padding_mask = create_masks(inp, tar)

         return predict_tokens
```

모델 로스 정의

```
loss_object = tf.keras.losses.SparseCategoricalCrossentropy(
    from_logits=True, reduction='none')

train_accuracy = tf.keras.metrics.SparseCategoricalAccuracy(name='accuracy')

def loss(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    loss_ = loss_object(real, pred)

    mask = tf.cast(mask, dtype=loss_.dtype)
    loss_ *= mask

    return tf.reduce_mean(loss_)

def accuracy(real, pred):
    mask = tf.math.logical_not(tf.math.equal(real, 0))
    mask = tf.expand_dims(tf.cast(mask, dtype=pred.dtype), axis=-1)
    pred *= mask
    acc = train_accuracy(real, pred)

    return tf.reduce_mean(acc)
```

학습

```
# overfitting을 막기 위한 ealrystop 추가
earlystop_callback = EarlyStopping(monitor='val_accuracy', min_delta=0.0001, patience=10)
# min_delta: the threshold that triggers the termination (acc should at least improve 0.0001)
# patience: no improvment epochs (patience = 1, 1번 이상 상승이 없으면 종료)

checkpoint_path = DATA_IN_PATH + model_name + '/weights.h5'
checkpoint_dir = os.path.dirname(checkpoint_path)

# Create path if exists
if os.path.exists(checkpoint_dir):
    print("{} -- Folder already exists \n".format(checkpoint_dir))
else:
    os.makedirs(checkpoint_dir, exist_ok=True)
    print("{} -- Folder create complete \n".format(checkpoint_dir))


cp_callback = ModelCheckpoint(
    checkpoint_path, monitor='val_accuracy', verbose=1, save_best_only=True, save_weights_only=True)
```

```
model = Transformer(**kargs)
model.compile(optimizer=tf.keras.optimizers.Adam(1e-4),
              loss=loss,
              metrics=[accuracy])
```

```
history = model.fit([index_inputs, index_outputs], index_targets,
                    batch_size=BATCH_SIZE, epochs=EPOCHS,
                    validation_split=VALID_SPLIT, callbacks=[earlystop_callback, cp_callback])
```

```
Epoch 1/30
5320/5320 [==============================] - 911s 171ms/step - loss: 0.8180 - accuracy: 0.8934 - val_loss: 1.2

Epoch 00001: val_accuracy improved from 0.89252 to 0.89281, saving model to /content/drive/MyDrive/Kaggle/챗봇
Epoch 2/30
5320/5320 [==============================] - 886s 167ms/step - loss: 0.8301 - accuracy: 0.8929 - val_loss: 1.2

Epoch 00002: val_accuracy did not improve from 0.89281
Epoch 3/30
5320/5320 [==============================] - 877s 165ms/step - loss: 0.8001 - accuracy: 0.8929 - val_loss: 1.2

Epoch 00003: val_accuracy improved from 0.89281 to 0.89284, saving model to /content/drive/MyDrive/Kaggle/챗봇
Epoch 4/30
5320/5320 [==============================] - 878s 165ms/step - loss: 0.7697 - accuracy: 0.8929 - val_loss: 1.3

Epoch 00004: val_accuracy improved from 0.89284 to 0.89299, saving model to /content/drive/MyDrive/Kaggle/챗봇
Epoch 5/30
5320/5320 [==============================] - 874s 164ms/step - loss: 0.7396 - accuracy: 0.8931 - val_loss: 1.3

Epoch 00005: val_accuracy improved from 0.89299 to 0.89322, saving model to /content/drive/MyDrive/Kaggle/챗봇
Epoch 6/30
5320/5320 [==============================] - 875s 164ms/step - loss: 0.7122 - accuracy: 0.8934 - val_loss: 1.3

Epoch 00006: val_accuracy improved from 0.89322 to 0.89355, saving model to /content/drive/MyDrive/Kaggle/챗봇
Epoch 7/30
5320/5320 [==============================] - 874s 164ms/step - loss: 0.6870 - accuracy: 0.8937 - val_loss: 1.3

Epoch 00007: val_accuracy improved from 0.89355 to 0.89394, saving model to /content/drive/MyDrive/Kaggle/챗봇
Epoch 8/30
5320/5320 [==============================] - 873s 164ms/step - loss: 0.6663 - accuracy: 0.8942 - val_loss: 1.3

Epoch 00008: val_accuracy improved from 0.89394 to 0.89436, saving model to /content/drive/MyDrive/Kaggle/챗봇
```

베스트 모델 불러오기

```
DATA_OUT_PATH = './data_out/'
SAVE_FILE_NM = 'weights.h5'

model.load_weights(os.path.join(DATA_IN_PATH, model_name, SAVE_FILE_NM))
```

결과 확인

```
char2idx = prepro_configs['char2idx']
idx2char = prepro_configs['idx2char']
```

```
text = "남자친구 승진 선물로 뭐가 좋을까?"
test_index_inputs, _ = enc_processing([text], char2idx)
outputs = model.inference(test_index_inputs)

print(' '.join([idx2char[str(o)] for o in outputs]))
```

이 방법도 시간이 너무 오래걸리고 집 와이파이 상황이 좋지 않아 Colab에서 돌리는 것이 무리였다. 하지만 확실히 Seq-to-Seq보다는 학습 속도가 빠른 것을 확인하였다.