IAB 코딩 기술서(Chat-bot with Transformer)

공과대학 기계항공공학부

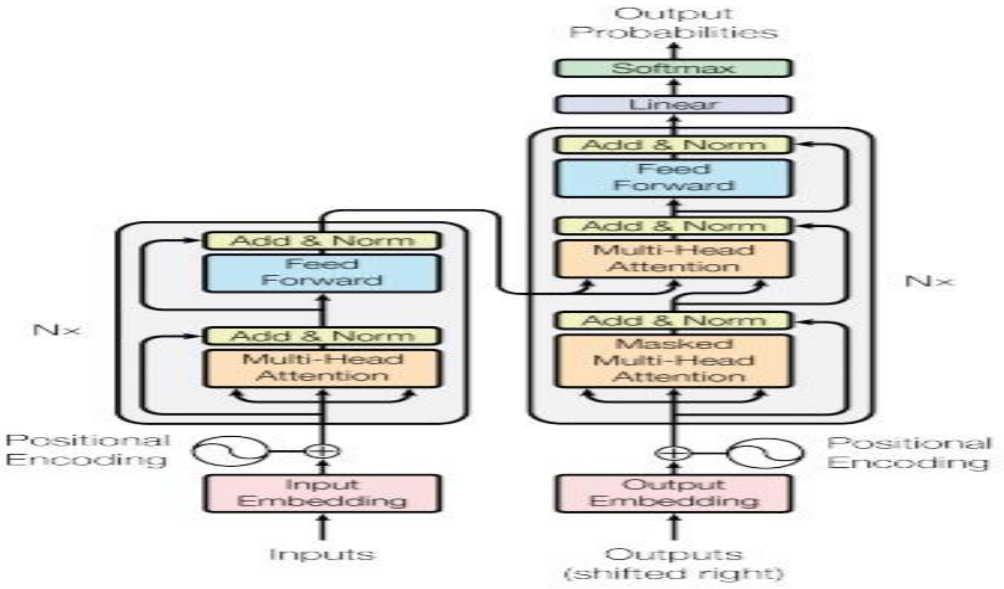
2016-18878 이종윤

1. 주제 선정 이유

인공지능은 앞으로 점점 더 발전해서 사람들이 해야 하는 귀찮거나 위험한 일들을 대신해줄 것이라 생각된다. 그런 인공지능 기술들이 우리 일상생활 속에 잘 녹아들기 위해서는 NLP(자연어 처리)가 잘 이뤄져야 할 것으로 생각됐습니다. 그래서 자연어가 처리되어 학습모델이 이해하는 과정을 구체적으로 알아보고 싶었기 때문에 Attention is all you need라는 논문에서 제시한 요즘 자연어, 이미지를 처리하는데 주요하게 사용되는 Transformer를 이용하여 한국어로 된 text로 간단한 대화가 가능한 Chat-Bot을 구현해보기로 결정했습니다.

2. Transformer

Attention is all you need에서 처음 제시한 학습모델로 Encoder-Decoder 구조를 따르면서도 attention이라는 개념을 추가시켜 어떤 문장이 입력으로 주어졌을 때 한 단어를 기준으로 다른 단어들 간의 관계를 주요 정보로 인지하여 그 정보들을 토대로 입력한 문장에 알맞은 문장을 출력하게 됩니다. Attention is all you need에서는 원래 문장을 번역해주는 기능을 위해 만들어 졌지만 attention이라는 개념이 유용하고 획기적이었기 때문에 현재는 앞에서도 언급했듯이 자연어 처리뿐만 아니라 이미지 프로세싱에도 사용이 되고 더 다양한 방식으로 사용 가능할 것으로 생각됩니다.



(Reference: Attention is all you need)

우선 트랜스포머는 입력의 단어들의 위치에 따라서도 의미가 달라지기 때문에 단어들이 임베딩된 벡터들이 위치정보도 가지도록 해줘야 하기 때문에 다음과 같은 코드로 PositionalEncoding을 해준다.

```
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_config(self):
    config = super().get_config().copy()
    config.update({
        'pos_encoding' : self.positional_encoding(position, d_model)
    })
    return config

  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)

    sines = tf.math.sin(angle_rads[:, 0::2])

    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, ...]

    return tf.cast(pos_encoding, tf.float32)

  def call(self, inputs):
    return inputs + self.pos_encoding[:, :tf.shape(inputs)[1], :]
```

그리고 attention 값들을 계산해주기 위해서 내적을 이용해 계산해주는 함수인 Scaled Dot Product Attention 코드는 다음과 같다.

```
def scaled_dot_product_attention(query, key, value, mask):

  matmul_qk = tf.matmul(query, key, transpose_b=True)

  depth = tf.cast(tf.shape(key)[-1], tf.float32)
  logits = matmul_qk / tf.math.sqrt(depth)

  if mask is not None:
    logits += (mask * -1e9)

  attention_weights = tf.nn.softmax(logits, axis=-1)

  output = tf.matmul(attention_weights, value)

  return output, attention_weights
```

여기서 Query는 영향을 받는 단어, Key는 영향을 주는 단어, Value는 영향에 대한 가중치라고 간단히 생각할 수 있다.

받아진 문장들에 대해 패딩을 마스킹해서 단어들이 Query가 Key를 고려할 때 padding있는 곳은 attention을 하지 않기 위해서 다음과 같은 함수를 구현합니다.

```
def create_padding_mask(x):
  mask = tf.cast(tf.math.equal(x, 0), tf.float32)
  return mask[:, tf.newaxis, tf.newaxis, :]
```

디코더의 셀프어텐션에서 한 단어를 판단할 때 미래의 단어를 고려하지 않도록 마스크를 씌우는 Look-ahead mask를 다음과 같이 구현할 수 있다.

```
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)
  return tf.maximum(look_ahead_mask, padding_mask)
```

앞에서 구한 가중치들을 dense layer를 통과시켜 얻어진 값들을 계산하는 MultiHeadAttention 코드는 다음과 같다.

```
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

    self.depth = d_model // self.num_heads

    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)

    self.dense = tf.keras.layers.Dense(units=d_model)

  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]

    query = self.query_dense(query)
    key = self.key_dense(key)
    value = self.value_dense(value)

    query = self.split_heads(query, batch_size)
    key = self.split_heads(key, batch_size)
    value = self.split_heads(value, batch_size)

    scaled_attention, _ = scaled_dot_product_attention(query, key, value, mask)

    scaled_attention = tf.transpose(scaled_attention, perm=[0, 2, 1, 3])

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

    outputs = self.dense(concat_attention)

    return outputs
```

Encoder와 Decoder를 구현한 코드는 다음과 같습니다.

Encoder_Layer, Decoder_Layer는 여기서는 하나의 Encoder, Decoder를 의미한다고 볼 수 있습니다.

```
def encoder_layer(dff, d_model, num_heads, dropout, name="encoder_layer"):
  inputs = tf.keras.Input(shape=(None, d_model), name="inputs")
  padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")
  attention = MultiHeadAttention(
      d_model, num_heads, name="attention")({
          'query': inputs, 'key': inputs, 'value': inputs,
          'mask': padding_mask
      })

  attention = tf.keras.layers.Dropout(rate=dropout)(attention)
  attention = tf.keras.layers.LayerNormalization(
      epsilon=1e-6)(inputs + attention)

  outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention)
  outputs = tf.keras.layers.Dense(units=d_model)(outputs)

  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 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")

  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')

  attention1 = MultiHeadAttention(
      d_model, num_heads, name="attention_1")(inputs={
          'query': inputs, 'key': inputs, 'value': inputs,
          'mask': look_ahead_mask
      })

  attention1 = tf.keras.layers.LayerNormalization(
      epsilon=1e-6)(attention1 + inputs)

  attention2 = MultiHeadAttention(
      d_model, num_heads, name="attention_2")(inputs={
          'query': attention1, 'key': enc_outputs, 'value': enc_outputs,
          'mask': padding_mask
      })

  attention2 = tf.keras.layers.Dropout(rate=dropout)(attention2)
  attention2 = tf.keras.layers.LayerNormalization(
      epsilon=1e-6)(attention2 + attention1)

  outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention2)
  outputs = tf.keras.layers.Dense(units=d_model)(outputs)

  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)
```

앞에서 구현한 Layer들을 num_layer만큼 쌓아줄 Encoder, Decoder 코드는 다음과 같다.

```
def encoder(vocab_size, num_layers, dff,
            d_model, num_heads, dropout,
            name="encoder"):
  inputs = tf.keras.Input(shape=(None,), name="inputs")
  padding_mask = tf.keras.Input(shape=(1, 1, None), name="padding_mask")
  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)

  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 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')

  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')
  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)

  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)
```

마지막으로 위의 코드들로 만들어진 Transformer이다.

```
def transformer(vocab_size, num_layers, dff,
                d_model, num_heads, dropout,
                name="transformer"):
  inputs = tf.keras.Input(shape=(None,), name="inputs")
  dec_inputs = tf.keras.Input(shape=(None,), name="dec_inputs")
  enc_padding_mask = tf.keras.layers.Lambda(
      create_padding_mask, output_shape=(1, 1, None),
      name='enc_padding_mask')(inputs)
  look_ahead_mask = tf.keras.layers.Lambda(
      create_look_ahead_mask, output_shape=(1, None, None),
      name='look_ahead_mask')(dec_inputs)
  dec_padding_mask = tf.keras.layers.Lambda(
      create_padding_mask, output_shape=(1, 1, None),
      name='dec_padding_mask')(inputs)
  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])
  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])
  outputs = tf.keras.layers.Dense(units=vocab_size, name="outputs")(dec_outputs)
  return tf.keras.Model(inputs=[inputs, dec_inputs], outputs=outputs, name=name)
```

3. 결과

Attention is all you need에서 제시한 모델의 hyperparameter들은 Encoder, Decoder의 입력과 출력 dimension(d_model)은 512이고 Encoder, Decoder(num_layers)는 각각 6개씩 쌓아 올렸고 attention head(num_heads)는 8개, hidden layer(d_ff)의 dimension은 2048로 제시하여 학습시켰기 때문에 동일한 구성으로 11800개의 한국말로 된 문장을 형태소까지 분류시키도록 하여 학습시킨 결과 accuracy는 0.14정도로 나왔지만 기본적인 '안녕하세요'같은 간단한 문장에도 제대로 답변을 하지 못하는 결과가 나왔습니다. 아무래도 학습시킨 데이터는 너무 적은데 비해 모델의 구조가 너무 복잡하기 때문에 overfitting되어 그런 결과가 나타났다고 생각됩니다. 그래서 앞에서 언급한 hyperparameter들을 계속 바꿔보면서 accuracy들을 비교해본 결과, Encoder, Decoder의 입력과 출력 Dimension은 128로 각각 4개씩 쌓아올리고 attention head는 8개, 내부 hidden layer는 256으로 설정해주었을 때 결과가 그나마 괜찮게 나왔었습니다. 아래 사진은 epoch을 50으로 설정하고 학습시킨 결과입니다.

```
Epoch 40/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0374 - accuracy: 0.1650
Epoch 41/50
185/185 [==============================] - 26s 142ms/step - loss: 0.0355 - accuracy: 0.1654
Epoch 42/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0344 - accuracy: 0.1658
Epoch 43/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0315 - accuracy: 0.1666
Epoch 44/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0303 - accuracy: 0.1669
Epoch 45/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0295 - accuracy: 0.1671
Epoch 46/50
185/185 [==============================] - 26s 142ms/step - loss: 0.0277 - accuracy: 0.1675
Epoch 47/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0262 - accuracy: 0.1679
Epoch 48/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0257 - accuracy: 0.1681
Epoch 49/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0245 - accuracy: 0.1686
Epoch 50/50
185/185 [==============================] - 26s 141ms/step - loss: 0.0228 - accuracy: 0.1689
```

Accuracy가 사실 0.17도 안되는 아주 안 좋은 결과라고 볼 수 있는데 epoch을 늘려도 accuracy는 크게 증가하지 않았기 때문에 그냥 50회까지만 학습시키고 테스트해본 결과는 다음과 같습니다.

```
==========================================================================================================
User: 안녕하세요
Home Assistant: 안녕하세요 .
==========================================================================================================

==========================================================================================================
User: 오늘 하루도 알차게 보냈습니다
Home Assistant: 정신이 없겠네요 .
==========================================================================================================

==========================================================================================================
User: 피곤한 하루 입니다.
Home Assistant: 푹 쉬세요 .
==========================================================================================================
```

```
==========================================================================================================
User: 오늘은 행복한 하루입니다
Home Assistant: 상대방도 미소짓게 해주세요 .
==========================================================================================================

==========================================================================================================
User: 치킨이 먹고싶다
Home Assistant: 에휴 .
==========================================================================================================
```

여러 번 테스트해봤지만 괜찮은 결과들만 가져온 것이고 0.16이라는 accuracy에 비해서는 생각보다 결과가 좋은 것으로 보입니다. 사실은 accuracy를 측정하는 것이 어떻게 보면 의미가 없을 수도 있습니다. 학습시킨 데이터에 그대로 출력하면 accuracy가 100%겠지만 그게 목적이 아니고 모델이 말들을 배워서 사람들이 하는 물음이나 말에 적절한 답변을 하기만 하면 되기 때문에 accuracy는 낮아질 수 있어도 비슷한 의미를 가진 문장만 출력한다면 문제가 되지 않다고 생각합니다. Attention is all you need에서도 다른 측정치를 사용하긴 했지만 accuracy가 거의 30~40에 정도로 그렇게 높지 않았습니다. 그리고 Chat-Bot이 인풋에 있는 단어들 중 특정한 단어집단안의 단어가 50%이상 포함되면 날씨를 알려주는 기능과 끝말잇기를 할 수 있도록 만들어주는 기능을 구현해본 결과는 아래와 같습니다.

```
==========================================================================================================
User: 끝말잇기 하자
좋습니다. 먼저 시작하세요!
끝말잇기 시작----------------------------------------------------------------------

 초고수 끝말잇기 ver.0.1
 Ctrl+Z를 입력하면 기권할 수 있습니다.

--------------------------------------------------

 1라운드를 시작합니다. 현재 0승 0패

 CPU : 차아할로겐산

 YOU : 산토끼

 CPU : 끼리끼리

 YOU : 리어카

 CPU : 카르바미노헤모글로빈

 YOU : 빈잔
 [오류] 사전에 없는 단어입니다.

 YOU : 빈곳
 [오류] 사전에 없는 단어입니다.

 YOU : 빈둥지
 [오류] 사전에 없는 단어입니다.

==========================================================================================================
User: 오늘 날씨 어때?
오늘 서울의 온도는 28.9도이고 습도는 70%입니다.
==========================================================================================================
```

```
==========================================================================================
User: 날씨 알려줘
오늘 서울의 온도는 28.9도이고 습도는 70%입니다.
==========================================================================================

==========================================================================================
User: 지금 날씨 알려줘
오늘 서울의 온도는 28.9도이고 습도는 70%입니다.
==========================================================================================
```

사실 위의 기능들을 구현하는데 별로 학습이라 할만한게 없었기 때문에 결과만 제시하겠습니다.

그리고 과연 이 모델이 학습되지 않은 데이터가 입력으로 들어왔을 때 잘 처리하는 지 확인한 결과는 다음과 같습니다.

```
a = input()
if a in questions:
  print('True')
else:
  print('False')
```

```
오늘 하루도 알차게 보냈습니다
False
```

```
==========================================================================================
User: 오늘 하루도 알차게 보냈습니다
Home Assistant: 정신이 없겠네요 .
==========================================================================================
```

```
a = input()
if a in questions:
  print('True')
else:
  print('False')
```

```
낮잠 좀 자야겠다
False
```

```
==========================================================================================
User: 낮잠 좀 자야겠다
Home Assistant: 안녕히 주무세요 .
==========================================================================================
```

```
a = input()
if a in questions:
  print('True')
else:ㄴ
  print('False')
```

```
내일 저녁에 뭘 먹을까?
False
```

```
==========================================================================================
User: 내일 저녁에 뭘 먹을까?
Home Assistant: 한 번만 더 연락해보는 건 어떨까요 .
==========================================================================================
```

거의 제대로 답변을 못했지만 '낮잠 좀 자야겠다'는 말에는 제대로 답변을 한 것을 확인할 수 있었습니다.

다음으로는 반대로 학습되지 않은 문장들을 출력할 수 있는지 확인해보았고 결과는 다음과 같습니다.

```
==========================================================================================
User: 내일 저녁에 뭘 먹을까?
Home Assistant: 한 번만 더 연락해보는 건 어떨까요 .
==========================================================================================
```

```
a = input()
if a in answers:
  print('True')
else:
  print('False')
```

```
한번만 더 연락해보는 건 어떨까요 .
False
```

학습되지 않은 말들 또한 잘 만들어 내는 것으로 보이지만 사실 문맥을 제대로 이해하지 못하고 답변을 했거나 답변 자체가 문법적으로 잘못된 경우가 있었습니다. 그리고 학습되지 않은 입력에 보통 학습되지 않은 출력을 하는 것으로 보였고 사실 이게 가능하도록 하는 것이 목적이었기 때문에 좋은 결과를 얻어 냈다고 할 수 없습니다. 데이터셋, 모델 모두 대화라는 광범위하는 주제를 모두 커버하기에는 작게 설정하여 이런 결과가 도출되었다고 생각됩니다.

4. 개선방안

학습시킨 데이터가 우울한 사람들을 위로해주는 말들로 구성된 데이터셋이었기 때문에 그와 관련된 물음에는 잘 답변을 하는 것으로 보였지만 그렇지 않은 데이터들에는 잘 대답을 못하는 것으로 보였습니다. 어찌보면 알지 못하는 단어들에 제대로 답변을 못하는 것은 당연하다고 생각됩니다. 따라서 거의 모든 대화 주제에 응답할 수 있도록 데이터셋을 아주 많이 늘려주고 그에 맞게 모델을 복잡하게 구성시켜 준다면 더 좋은 결과를 출력할 수 있을 것으로 생각됩니다. 하지만 그렇게 하기까지는 지금의 상황으로는 학습데이터를 구하기 어렵다는 점과 모델을 복잡하게 구성시켜줬을 때 현재 가지고 있는 컴퓨터의 기능상의 한계로 어려움이 있을 것으로 생각됩니다.

5. 참고자료

Attention is all you need(Google, Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, Illia Polosukhin)

딥 러닝을 이용한 자연어 처리 입문 (https://wikidocs.net/31379, https://wikidocs.net/89786)

크롤링을 이용한 끝말잇기 구현 (https://m.blog.naver.com/njw1204/221364710539)