16. RNN과 어텐션을 사용한 자연어 처리

비주얼 어텐션(Visual Attention)

트랜스포머(attention is all you need)

위치 인코딩(Positional Encoding)

2021.05.16
고은별

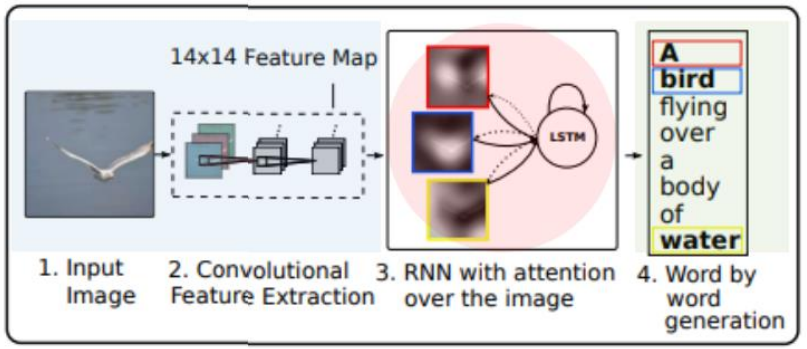
16.4.1 비주얼 어텐션

: 이미지를 입력 데이터로 사용하여 이미지를 설명하는 문장을 생성하는 Image Captionaing 문제에서 사용됨.

input

"A woman is laughing."

output



Encoder – Feature Extraction

Convolutional Neural Network 모델으로 이미지의 특징을 잘 요약하는 feature map 생성

Attention

Decoder에서 단어를 출력할 때 단어와 가장 유사한 feature map과의 attention score을 계산

Decoder – Word Generation

이전 시점에 출력된 단어의 정보, feature map, attention score를 decoder의 input으로 사용하여 순차적으로 단어를 출력

16.4.2 Transformer (Attention is all you need)

Transformer Network의 등장 배경

: Attention is All You Need(2017) 에서 발표된 기계번역 신경망. Self-Attention기법을 활용해 기존의 전통적인 RNN과 CNN 계열의 신경망 구조를 탈피하려고 한 것이 특징.

CNN – Focus on the local context → Weak for Long-Distance

RNN – Factorization → Dependency Problem for Long-Distances

2021년 기준 고성능 모델 -> based on Transformer architecture

GRT – Transformer의 Decoder 활용

BERT – Transformer의 Encoder 활용

16.4.2 Transformer (Attention is all you need)

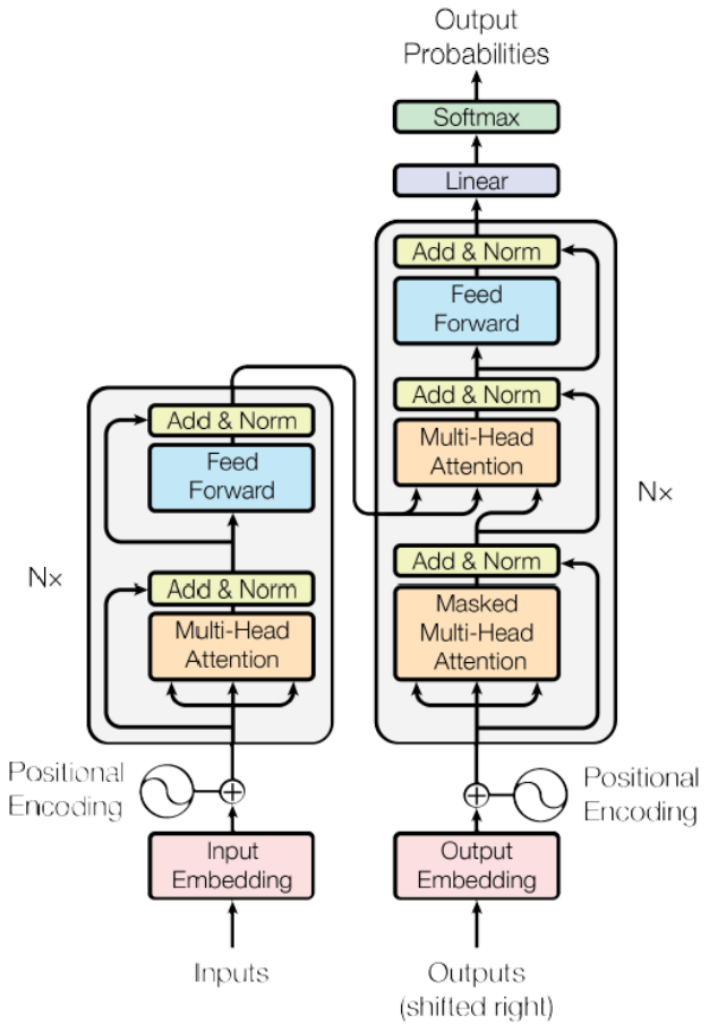


Figure 1: The Transformer - model architecture.

위치 인코딩(Positional Encoding)

멀티헤드어텐션(Multi-head Attention)
셀프어텐션(Self Attention)

1) 인코더(Encoder)

- input으로 Embedding 후 token별로 나눠진 sentence를 입력받는다.
- 위치 인코딩 이후 부분을 N번 쌓아올린다.(논문에서는 N=6)

2) 디코더(Decoder)

- 훈련하는 동안 target sentence를 input으로 받는다.
- Encoder의 출력을 받음. 똑같이 N번 쌓아올린다.
- 매 Timestep마다 다음 단어에 대한 확률을 출력한다.

16.4.2 Transformer – 위치 인코딩(Positional Encoding)

Positional Encoding의 기본 메커니즘

- 언어를 이해하는 데 어순은 매우 중요한 역할을 하므로 이 정보에 대한 처리가 필요.
- RNN과 달리 입력을 순차적으로 주지 않는다. 따라서 시퀀스 정보를 넣어줘야 하는 문제가 생김.
- Embedding 된 input과 같은 크기의 벡터를 각각에 더해줌으로써 각 단어의 상대적인 위치정보에 대해서 알려주는 것.

$$PE_{(pos,\,2i)} = sin(pos/10000^{2i/d_{model}})$$

$$PE_{(pos,\,2i+1)} = cos(pos/10000^{2i/d_{model}})$$

```
def positional_encoding(model_dim, sentence_len):
      encoded_vec = np.array([pos/np.power(10000, 2*i/model_dim)
                              for pos in range(sentence_len) for i in range(model_dim)])
  encoded_vec[::2] = np.sin(encoded_vec[::2])
  encoded_vec[1::2] = np.cos(encoded_vec[1::2])
  # 텐서플로 그래프에 올리기위해 constant로 만들며, [시퀀스 길이*피처크기]의 행렬을 생성한다.
  return tf.constant(encoded_vec.reshape([sentence_len, model_dim]), dtype=tf.float32)
```