# 어텐션 구조와 어텐션 메커니즘


과목명 | 딥러닝 프레임워크
담당교수 | 최인엽 교수님
팀원 | 202004245 한지은, 202184028 성윤주
제출일 | 2024.05.04

**[Attention이란?]**

Attention 메커니즘이 처음 소개된 것은 2015년에 발표된 'NEURAL MACHINE TRANSLATION BY JOINTLY LEARNING TO ALIGN AND TRANSLATE' 이라는 논문을 통해서이다. 이 논문에서 Attention 메커니즘은 소스 문장의 모든 정보를 고정된 길이의 벡터로 압축하는 기존의 Seq2Seq 모델의 한계를 넘어서, 정보 손실을 최소화하는 방식으로 소개되었다. 이를 통해 모델은 문장의 어느 부분에 더 집중해야 할지를 동적으로 결정할 수 있게 되었고, 결과적으로 번역의 정확도와 효율성이 크게 향상되게 된다. Attention의 기본 아이디어는 디코더에서 출력 단어를 예측하는 매 시점마다, 인코더에서의 전체 입력 문장을 다시 한 번 참고한다는 점이다. 이 때 전체 입력 문장을 전부 다 동일한 비율로 참고하는 것이 아니라, 해당 시점에서 예측해야할 단어와 연관이 있는 입력 단어 부분을 좀 더 집중해서 보게 된다.

Attention 메커니즘은 특정 데이터 요소의 중요성을 결정하는 기법으로, 출력이 특정 입력에 얼마나 주목해야 하는지를 학습한다. Attention 구조는 Attention 메커니즘을 기반으로 한 신경망 구조로, 이러한 구조는 입력 데이터의 전체적인 관계를 병렬적으로 분석하여 처리한다.

**[Attention의 장점]**

RNN은 짧은 참조 윈도우 크기를 가지고 있기에 입력이 길어지면 주어진 시퀀스보다 긴 입력을 고려하지 못하는 문제를 가지고 있다. 기존 RNN, LSTM, GRU 등의 Seq2Seq 모델은 시퀀스의 초기 부분에 더 주목하게 되며, 입력이 길어질수록 시퀀스 뒷부분의 정보를 손실하기 쉽다. 또한 Seq2Seq은 병렬화 문제와 Long Distance Dependency 문제를 가지고 있다. 먼저 병렬화 문제는 Seq2Seq가 구조상 순차적으로 입력을 처리해야 하기에 야기된다. 이 때문에 대규모 데이터셋의 경우 학습 시간이 지나치게 길어졌다. 다음으로 Long Distance Dependency 문제는 참조 윈도우의 크기가 고정이기 때문에 시퀀스에서 멀리 떨어진 항목간의 관계성은 Gradient Vanishing/Exploding 문제로 학습이 원활하게 되지 않는 문제이다.

기존의 RNN 계열에서 순차적이라는 것은 하나의 장점이었지만, 동시에 한계를 갖는다. 가까운 요소에 대해서만 의미를 강하게 가질 수 있기 때문이다. 하지만, Attention을 통하여 멀리 있는 요소까지도 연관성(유사도)을 크게 가질 수 있다. 이렇게 Attention은 "Attention is all you need"라는 논문을 통하여 병렬처리가 어려워 연산 시간이 오래 걸리는 RNN을 넘어 새로운 모델인 트랜스포머에 주요한 역할로써 사용되었다.

**[Attention 메커니즘]**

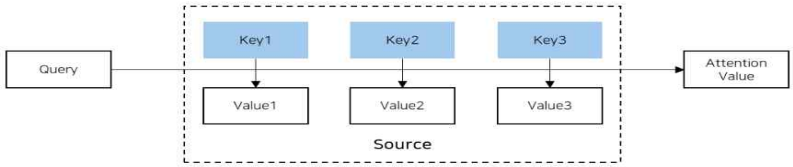
Attention은 Q(query), K(key), V(value) 3개의 벡터를 입력으로 받아 query와 key-value 쌍을 출력에 대응하는 것으로 설명될 수 있다. 각 값에 할당된 가중치는 해당 key에 대응되는 query에 대한 유사도로 계산된다. 자주 사용되는 유사도 계산 함수에는 dot product, splice, detector 등이 있다. 이렇게 얻은 Attention 가중치는 Softmax 함수를 사용하여 각 확률의 합이 1이 되도록 정규화한다. 이때 Softmax 함수란 로지스틱 함수를 다차원으로 일반화한 함수로, 주로 다항 로지스틱 분석에 사용되며 인공 신경망에서는 확률분포를 얻기 위한 마지막 활성화 함수로 주로 사용된다. 마지막으로 V에 대응하는 가중치들에 가중을 더하고 최종 Attention을 얻는다.

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$$

Attetion(Q,K,V)의 식은 위와 같다. 단 한 줄의 수식이지만 코드로 구현한다면 5개의 Layer(MatMul, Scale, Mask(Opt.), SoftMax, MatMul)를 거쳐 계산되게 된다.

**[Attention 응용]**

다양하게 응용된 Attention 매커니즘은은 Self Attention구조를 기본으로 두고 있다. 이에 따라서 Self Attention에 대해 좀 더 다뤄보겠다. Self Attention은 쿼리(query), 키(key), 밸류(value) 3개 요소 사이의 문맥적 관계성을 추출하는 과정이며 입력 벡터 시퀀스에 쿼리(Q), 키(K), 밸류(V)를 만들어주는 행렬(W)을 각각 곱하는 방식으로 이뤄진다. ($Q = X * W_Q$, $K = X * W_K$, $V = X * W_V$)




이는 인코더와 디코더 블록 모두에서 수행되며 세 가지 행렬은 태스크를 가장 잘 수행하는 방향으로 업데이트된다. 예를 들어, 주어 동사 목적어로 구성된 문장이 있다고 치면 인공지능은 학습을 할 때 이 문장을 각각의 단어로 쪼개어 그 단어가 주어인지 동사인지를 구분하기 위해 중요도를 측정하여 강조시킨다. 이때 사용되는 Self Attention Function은 입력의 시퀀스의 각 단어에 대한 가중치를 계산하는 함수이며 각 단어의 중요도를 측정하여 출력 결과에 반영한다.

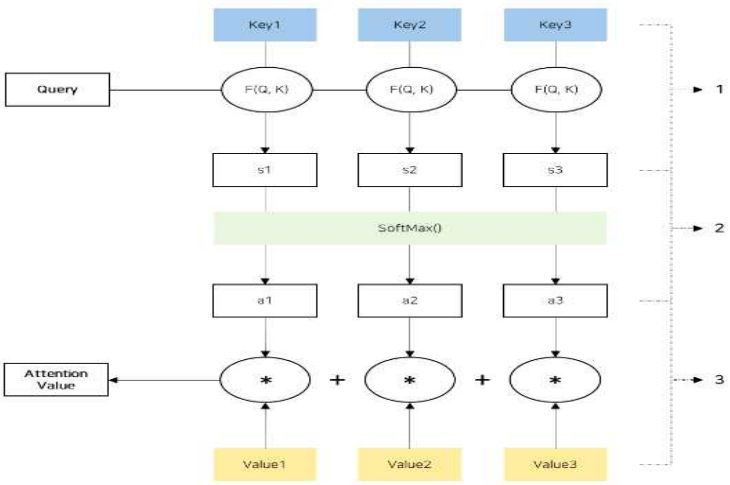
식은 위에 언급된 것과 같고 Q=query, K=key, V=value이며 query는 현재 출력 단어를 나타내는 벡터(t시점의 디코더 셀에서의 은닉상태)이고, key와 value는 입력 시퀀스의 각 단어에 대응하는 벡터(모든 시점의 인코더 셀의 은닉 상태)이다.

Attention Function은 다음의 순서로 계산이 이루어진다.

(1) query 벡터와 key 벡터간의 유사도(Attention Score)를 측정한다.

(2) 유사도를 정규화하여 각 단어의 가중치(인코더의 각 시점의 정보가 얼마나 중요한지 표현되는 지표)를 계산한다.

(3) (2)의가중치를 value 벡터와 곱한 뒤 모든 인코더 시점을 합산해 Attention 값을 구한다.



또한 Attention은 어텐션 스코어의 계산방법과 구조에 따라 여러 메커니즘으로 나뉜다.

1. Dot-Product Attention

Dot-Product Attention은 쿼리와 키의 내적을 사용하여 어텐션 스코어를 계산한다.

식: $Score(Q,K) = QK^T$,

2. Additive Attention

Additive Attention은 쿼리와 키를 합친 후 학습 가능한 가중치 행렬을 적용한 뒤 활성화 함수를 적용하여 어텐션 스코어를 구한다.

식: $Score(Q,K) = v^T * tanh(W1 * Q + W2 * K)$

3. Multi-Head Attention

Multi-Head Attention은 여러개의 독립적인 어텐션을 병렬로 계산하여 결과를 결합한다. 각 어텐션 헤드는 Scaled Dot-Product Attention를 기반으로하며 서로 다른 가중치를 사용하여 여러 관점을 학습할 수 있다.

$$MultiHead(Q,K,V) = Concat(head_1, \ldots, head_h) * W^O$$

식: $$head_i = Attention(Q * W_i^Q, K * W_i^K, V * W_i^V)$$

4. Scaled Dot-Product Attention

Scaled Dot-Product Attention은 내적 값을 키의 차원 수의 루트로 나누어 각 차원에 대한 영향력이 과도하게 커지는 것을 방지하고 안정성을 향상시킨다.

다음은 Scaled Dot-Product Attention에 대해서 자세히 알아보겠다. Scaled Dot-Product Attention은 내적 값을 정규화하여 과도한 집중을 방지하고 모든 토큰에 대한 정보를 적절하게 반영한다. 다른 Attention보다 속도가 빠르며 병렬 처리에 용이하고 다양한 입력 시퀀스 길이를 처리할 수 있어 안정된 학습이 가능하여 머신의 성능을 향상시킨다는 장점이 있다. Scaled Dot-Product Attention의 스코어 계산 식은 아래와 같다.

$$Score(Q,K) = QK^T/\sqrt{dk}$$

위의 식은 query 벡터와 key 벡터 간의 유사도를 계산하는 식이다. 각각 Q=query, K=key, V=value 벡터를 나타내며 dk는 key 벡터의 차원 수를 의미한다. 루트 차원 수로 나눠주는 이유는 key 벡터의 차원 수가 증가할수록 내적 값이 커지기 때문이다.

내적은 두 벡터 간의 유사도를 측정하는 방법 중 하나로, 두 벡터가 얼마나 같은 방향을 향하고 있는지를 나타내며 두 벡터가 비슷한 방향을 향할수록 내적 값은 커진다. key 벡터의 차원 수가 증가하면 각 차원의 값이 변화하는 범위가 더 커지므로 벡터의 표현 공간이 더 크고 다양해진다. 이는 입력 시퀀스와 현재 쿼리 벡터 간의 유사도가 더 높게 나타날 수 있음을 의미한다.

Scaled Dot-Product Attention에서 유사도가 높다는 것은 강한 어텐션을 가지는 토큰에 더 많은 가중치가 부여된다는 의미인데 특정 토큰의 정보가 더 잘 추출되어 최종 결과에서 큰 영향을 미치는 문제가 발생한다. 내적 값이 클수록 해당 토큰에 더 많은 가중치가 부여되어야하지만 최종 결과가 특정 토큰의 값에 과도하게 집중되어 버리는 것은 바람직하지 않다. 따라서 Scaled Dot-Product Attention에서는 내적 값이 key 벡터의 차원 수에 비례하지 않도록 하기 위해서 내적 값을 루트 key 벡터의 차원 수로 나누어주는 작업을 수행한다. 이렇게 하면 어텐션 분포를 조절할 수 있고 어텐션 가중치가 과도하게 특정 토큰에 집중되는 것을 방지하여 모든 토큰에 대한 정보를 고르게 추출할 수 있다. 그 다음엔 softmax 함수를 통해 정규화

를 진행하여 어텐션 분포를 조절하고 어텐션 가중치를 균등하게 분배할 수 있도록 한다. 그리고 어텐션 가중치를 이용하여 입력 시퀀스에서 해당 토큰의 정보를 추출하기 위해 value를 곱하여 최종 값을 구하고 마지막으로 Attention 값과 디코더의 t시점의 은닉상태를 연결(Concatenate)하여 새로운 입력으로 사용하여 출력한다.

scaled dot-product attention의 코드는 다음과 같다.

```
def sacled_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
```

위에 서술된 내용 이외에도 유클리드 거리(euclidean distance), 맨하탄 거리(manhattan distance), 코사인 유사도(cosine similarity)등 유사도 계산방법에 따른 다양한 Attention 메커니즘이 구현되어있다. 이 메커니즘은 현재 자연어 처리 및 기계 번역과 같은 분야에서 중요한 역할을 하고 있으며 향후에도 이미지처리나 음석인식 분야 등 다양한 방면에서 활약할 수 있을 전망이다.