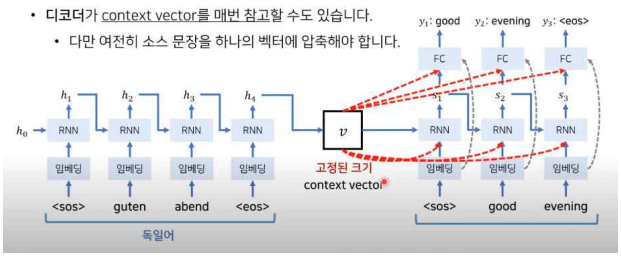
[Transformer: Attention Is All You Need 개념 정리]

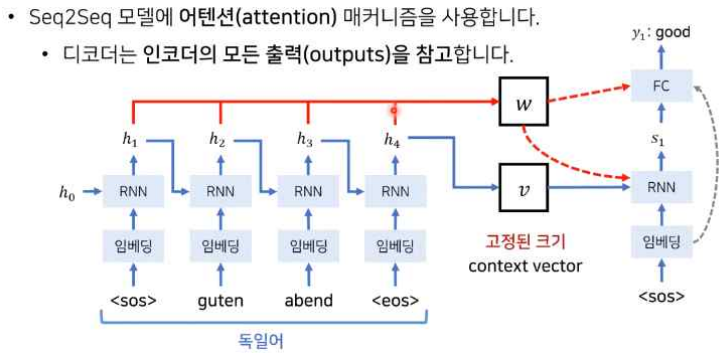
2021년 기준으로 최신 고성능 모델들은 Transformer 아키텍처를 기반 ->RNN에서 완전히 독립
GPT는 Transformer의 디코더 아키텍처를 활용하고 있고, BERT의 경우 Transformer의 인코더 아키텍처를 활용

1. 기존 Seq2Seq 모델들의 한계점

context 벡터 v에 소스 문장의 정보를 압축하게 된다 -> 병목이 발생하여 성능 하락의 원인이 됨
[문제 상황] 매번 단어가 입력이 될 때마다 hiddenstate 값이 갱신이 됨 => 단어가 입력될 때마다 이전까지의 입력에대한 정보가 담겨 있는 hidden state 값을 갱신 => 다양한 경우의 수를 포괄할 수 없게 됨
[문제 상황] 하나의 문맥 벡터가 소스 문장의 모든 정보를 가지고 있어야 하므로 성능이 저하
[해결 방안] 그렇다면 매번 소스 문장에서의 출력 전부를 입력으로 받으면? -> 최신 GPU는 많은 메모리와 빠른 병렬 처리를 지원하므로 불가능한 해결책이 아니게 됨



이와 같은 방식으로 context 벡터를 매번 RNN의 입력으로 넣어주면 조금 성능이 개선이 되겠지만, 여전히 모든 문장의 정보를 하나의 고정된 크기의 context 벡터에 압축해야한다는 점은 병목현상을 유발
=> 제안이 하나 나옴
=> 그럼 그냥 출력값을 하나의 context 벡터에 압축하지 말고, 모든 소스문장에서의 출력 전부를 압축 없이 전부 입력으로 넣으면 어떨까?
=> 최신 GPU와 많은 메모리가 해당 주장을 가능하게 함 -> 빠른 병렬 처리를 지원하기 때문



Seq2Seq 모델에 attention 메커니즘을 사용 : 디코더는 인코더의 모든 출력을 참고
hidden state 값이 매번 나올 때마다 하나의 배열에 이 값들을 저장해 둠, 따라서 매 단어가 들어왔을 때 갱신되는 hidden state 값들을 모두 기록함
이 값들을 참고해서 출력단어가 매번 생성될 때마다 소스문장 전체를 반영하겠다는 것

디코더 파트에서 매번 hidden state 값을 만들 때
=> 이전 단어의 loop에서 전달되는 이전 디코더 생성문에 대한 정보 + 소스 문장 전체에 대한 정보 가 전달됨
=> 그럼 이 두 정보를 대상으로 행렬곱을 수행하여 에너지값을 만들어냄 (어떤 단어에 집중, 즉 가중치를 두어야할지를 수치화해서 나타낸 값)
=> energy 값을 softmax를 취해서 확률값을 구한 뒤 소스문장 각각의 hiddenstate 값에 대해서 어떤 vector에 가중치를 두면 좋을지를 반영 -> 그 가중치값을 hiddenstate에 곱한 것을 각각의 비율에 맞게 더해줌 -> 그 weighted sum 값을 이제 decoder 생성에 매번 반영하게 되는 것

2. Seq2Seq with Attention

디코더는 매번 인코더의 모든 출력 중에서 어떤 정보가 중요한 지 계산
i = 현재의 디코더가 처리 중인 인덱스
j = 각각의 인코더 출력 인덱스
에너지 (Energy) = eij = a(si-1, hj)
=> 이전까지 내 디코더 출력값 si-1과 모든! 인코더 출력값을 의미하는 hj를 참고하여 이번 디코더 출력을 만들어내겠다는 뜻
가중치 (Weight) =

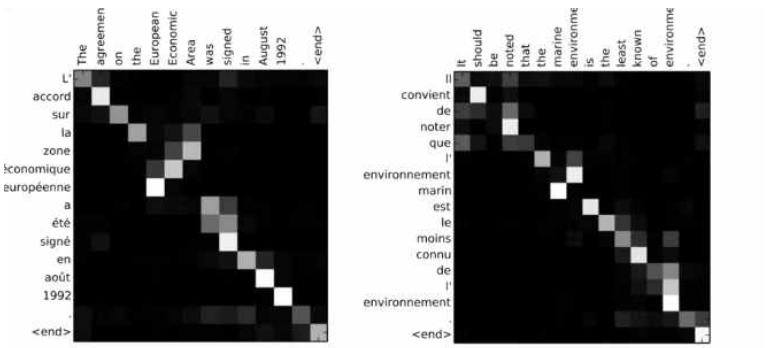
$$\alpha_{ij} = \frac{\exp(e_{ij})}{\sum_{k=1}^{T_x} \exp(e_{ik})}$$

그 에너지값에 소프트맥스를 취해서 비율을 만들어냄 => 즉, 어떤 h값과 가장 많은 연관을 가지고 있는지에 대한 척도를 소프트맥스 함수를 통해 생성해냄

$$c_i = \sum_{j=1}^{T_x} \alpha_{ij} h_j$$

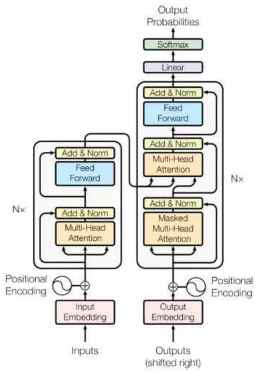
그리고 이러한 과정을 통해 생성된 에너지와 가중치의 곱 (weighted sum)을 디코더의 이번 입력값으로 넣어주겠다

Attention 기법의 장점 => 어텐션 시각화가 가능



++ 논문 리뷰 들어가기 전
- 2021년 기준으로 현대의 자연어 처리 네트워크에서 핵심이 되는 논문
- 트랜스포머는 RNN이나 CNN을 전혀 필요로 하지 않는다

3. 트랜스포머 Transformer

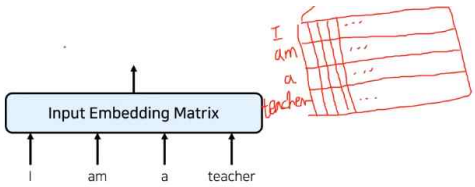


트랜스포머는 RNN이나 CNN을 전혀 사용하지 않음, 그러나 그렇게 된다면 문장 안의 단어들의 순서에 대한 정보를 주기가 어렵기 때문에 Positional Encoding을 활용함

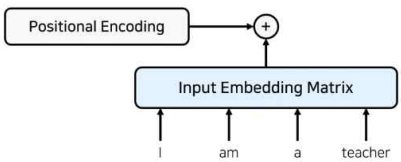
BERT와 같은 향상된 네트워크에서도 채택하고 있으며

인코더와 디코더로 구성되어 Attention 가정을 여러 레이어에서 반복하게 된다.
=> 즉 인코더가 N 번만큼 여러 번 중첩되어 사용하도록 만듦

(1) 트랜스포머의 동작 원리 : 입력값 임베딩 (Embedding)
임베딩을 진행하는 이유 -> 입력차원 자체는 특정 언어에서 존재할 수 있는 단어의 개수와 같음, 단어의 개수가 많을 뿐만 아니라 단어의 표현이 원핫인코딩으로 표현됨
-> 모델에 넣을 때는 임베딩을 거쳐서 조금 더 적은 차원의 continuous한 실수값으로 만드는 것

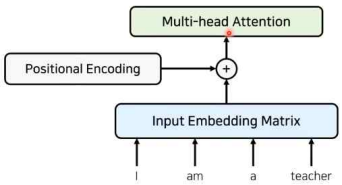


하나의 문장이 들어오게 되면 하나의 행 당 하나의 단어가 배치되고, 열은 임베딩 차원 (언어의 단어 개수)로 구성, 이는 모델러가 값을 지정할 수 있는데, 이 논문에서는 embed_dim을 512 정도로 한정함



이때 우리가 RNN 기반을 사용한다고 한다면 순서에 대한 정보를 각 hidden state가 가지고 있게 됨
그러나 RNN을 활용하지 않는 트랜스포머의 경우 **위치 정보를 포함하는 임베딩을 활용**해야 한다.
이를 위해 트랜스포머에서는 Positional Encoding을 활용 -> 인풋 임베딩 매트릭스와 동일한 차원을 가지는 위치 인코딩 벡터를 element-wise 덧셈을 통해 추가함

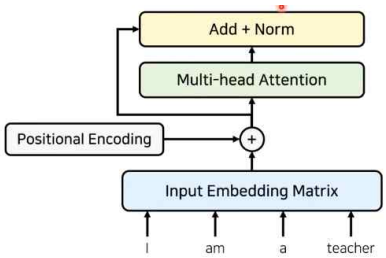
(2) 트랜스포머의 동작 원리 : Encoder



인코딩 이후 Attention을 진행 -> 즉 어텐션레이어의 인풋 값에는 **문장의 단어 정보와 위치 정보가 담긴 데이터**임
그리고 각각의 단어를 활용하여 Attention을 수행

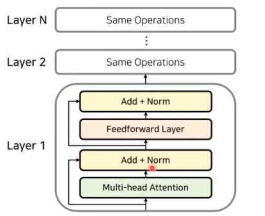
인코더 파트 Attention : **self-attention** ->각각의 단어가 **서로에게 어떤 연관성을 가지는지** 구하기 위해 사용
input 단어들 (I, am, a, teacher) 각각이 서로에게 얼마나 큰 영향을 주는지 attention 점수 scoring -> 단어 간의 연관성을 학습

즉, Attention은, 전반적인 입력 문장의 **문맥에 대한 정보**를 학습하도록 만드는 것이다.



ResNET과 같은 네트워크에서 활용되는 잔여 학습 (Residual Learning)을 성능 향상을 위해 활용
단순히 Layer을 거치면서 반복적으로 단순하게 값을 갱신하는 것이 아니라
특정 layer을 건너뛰어서 복사가 된 값을 그대로 넣어주는 기법 (Residual Connection을 통해 전달된 값)

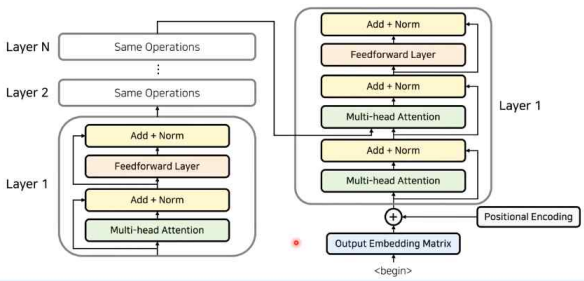
이를 통해 네트워크는 기존 정보를 residual connection을 통해 전달받으면서 추가적으로 잔여 attention값을 받게 되어 학습을 진행함
전반적인 학습 난이도가 낮음, 초기 모델 수렴 속도가 낮게 되어 global optimal에 도달할 확률이 크다



모델은 Attention과 정규화(Normalization) 과정을 반복하며,
각 레이어는 서로 다른 파라미터 값을 가지게 된다
즉, attention과 feedforward 에 활용되는 파라미터 값이 레이어마다 다르다

하나의 layer에서 나오는 값과 이어서 들어가는 값의 dimension은 동일하다

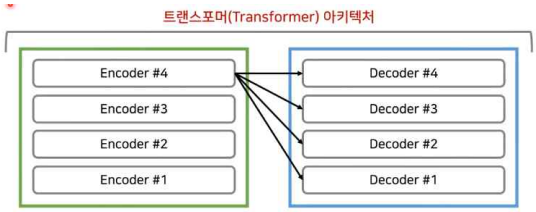
(3) 트랜스포머의 동작 원리 : Decoder



입력값이 들어온 뒤 여러 인코더 layer을 거친 뒤 나오게 되는 마지막 출력값은 디코더에 들어가게 됨
-> 이와 같이 해 주는 이유는 우리가 앞서 seq2seq모델의 attention 매커니즘을 활용했을 때와 동일하게, 디코더의 경우 매번 출력 시 입력 소스 문장 중에서 어떤 단어에 더 많이 초점을 맞춰야 할지 알려주기 위해서

디코더 파트도 마찬가지로 여러개의 layer을 거치게 되고, 마지막 layer의 출력값이 바로 실제로 우리가 수행한 결과
이때 각각의 layer은 encoder에서 출력된 값을 매번 디코더 레이어의 입력으로 받게 됨

참고로 하나의 디코더 레이어에서는 2개의 attention을 활용
첫 번째 Attention : 인코더 attention과 마찬가지로 각 단어 간의 연관성을 파악하는 attention -> 출력되고 있는 문장의 전반적인 표현을 학습
두 번째 Attention : 인코더에 대한 정보를 attention -> 출력되고있는 단어가 소스 문장의 어떤단어와 연관이 있는지 파악하는 attention -> encoder-decoder attention이라고 하기도 함

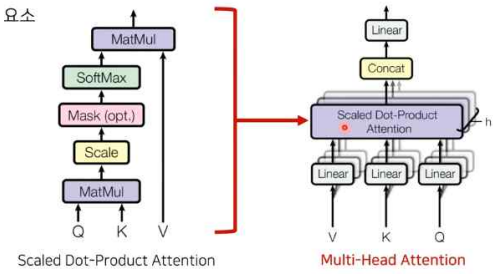


이와 같이 트랜스포머에서는 마지막 인코더 레이어의 출력이 모든 디코더 레이어의 두 번째 attention에 입력된다
통상적으로는 encoder layer의 layer 개수와 decoder layer의 개수를 맞춰줌

트랜스포머에서도 인코더와 디코더 구조를 따르게 됨
이때, RNN을 활용하지 않으며, 인코더와 디코더 layer을 다수 사용한다는 점이 특징
-> lstm이나 RNN의 경우 인코더를 고정된 크기로 사용하고, 입력 단어의 개수만큼 반복적으로 레이어를 걸쳐 hidden state 값을 만들어내었다면
-> transformer의 경우에는 문장을 한 번에 입력받아 한 번에 attention값을 받음, 즉 한 번에 입력을 받고 위치 정보는 따로 임베딩을 해서 더하여 병렬적으로 출력값을 구할 수 있게 됨

<eos> 토큰이 나올 때까지 디코더를 이용하게 된다

(4) 트랜스포머의 동작 원리 : Attention



인코더와 디코더는 Multi-Head Attention 레이어를 사용
어텐션의 실제 구조는 왼쪽의 사진과 같음

어텐션을 위한 세 가지 입력 요소
(1) 쿼리 (Query) : 어떤 관련성이 있는 지에 대한 질문을 물어보는 주체 ex) I
(2) 키(Key) : attention을 수행할 대상 (입력) ex) I am a teacher 각각의 단어
(3) 값(Value) : ex)

쿼리로 들어온 단어가 key의 각각의 단어들과 얼마나 연관성이 있는지 scoring을 matmul 및 scaling, softmax 함수를 통해 진행함
마지막으로 value와 곱함 -> 가중치가 적용된 출력값을 만들어낼 수 있게 되는 것

** 여기서 실제로 입력값이 들어왔을 때 **h개로 구분되게 만듦** -> **h개의 서로 다른 attention 컨셉을 학습**하도록 유도하는 과정임
더욱더 구분된, 다양한 특징을 학습하도록 함

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

$$head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

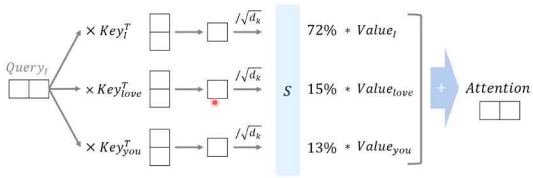
$$MultiHead(Q,K,V) = Concat(head_1, \dots, head_h)W^O$$

h: 헤드(head)의 개수

1번 식 : dk는 scale factor, 즉 스케일링을 해 주는 과정 -> 왜 하는지 생각해보면 softmax 함수의 특징을 고려하면 바로 알 수 있음, 0 근처에 값이 형성되게 되면 들쭉날쭉, gradient vaninshing 문제를 해결하기 위함

(5) 트랜스포머의 동작 원리 : (하나의 단어) 쿼리, 키, 값 (Query, Key, Value)

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

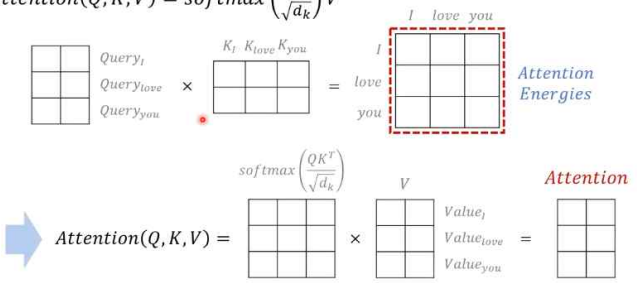


어텐션을 위해서는 쿼리, 키, 값 (Query, Key, Value)이 필요함
이는 각 단어의 임베딩 (Embedding)을 이용하여 생성이 가능하다
임베딩 차원 (d) -> Query, key value 차원 (d/h)

(6) 트랜스포머의 동작 원리 : (행렬) 쿼리, 키, 값 (Query, Key, Value)

실제로는 행렬(matrix) 곱셈 연산을 이용해 한꺼번에 연산이 가능하다

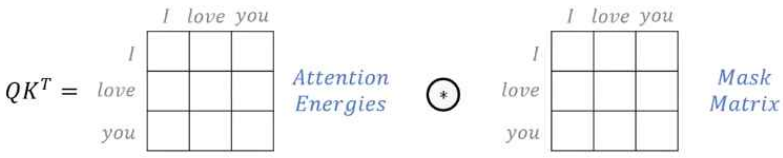
- $Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V$



Attention Energy값은 각 단어가 각 킷값에 얼마나 높은 중요 관련도를 부여했는지 구해놓은 행렬
softmax로 확률 값으로 변환
최종적으로 반환된 **Attention 행렬은 입력 차원과 동일하게 유지**되는 것을 확인할 수 있음

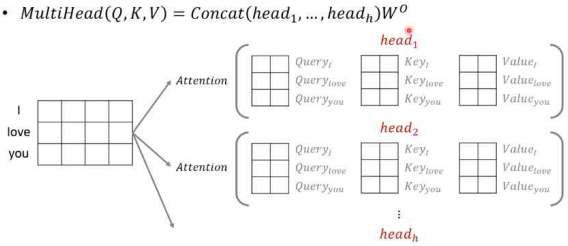
** 마스크 행렬의 활용
마스크 행렬 (mask matrix)를 이용해서 특정 단어는 무시할 수 있도록 설정이 가능하다
-> **Attention energy 행렬과 같은 크기의 mask matrix를 생성**하여 이를 **element-wise로 곱**해주며 이러한 연산 수행이 가능하도록 하는 것
-> 즉 어떤 특정 값을 참고하지 않도록 하고 싶을 때 사용하는 기법

마스크 값으로 **음수 무한의 값**을 넣어 softmax 함수의 출력이 0에 가까워지도록 설정

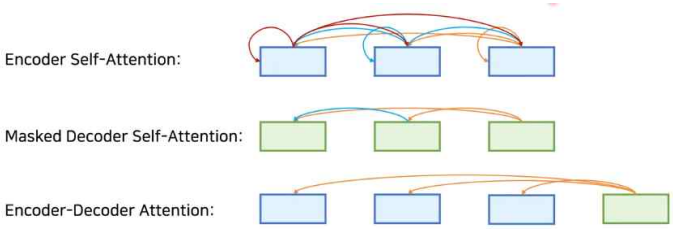


그리고 아래와 같이 concat 연산을 통해 차원을 맞추어주면 완성

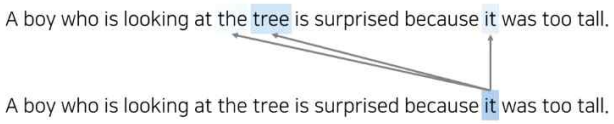
- $MultiHead(Q,K,V) = Concat(head_1,\dots,head_h)W^O$



** Attention의 종류
트랜스포머에서는 세 가지 종류의 어텐션 레이어가 활용된다



(1) Encoder Self-Attention

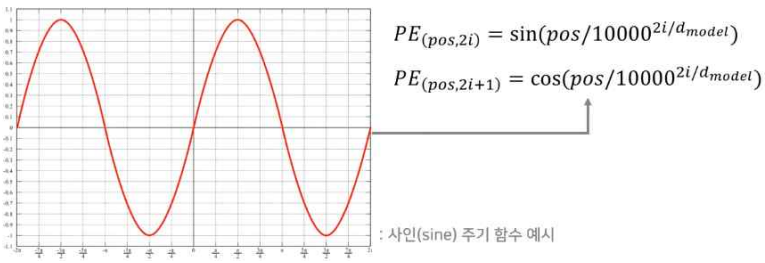


: 하나의 입력 문장 내의 단어들이 서로가 서로와 얼마나 큰 연관을 가지고 있는지
: 즉 문장 내의 연관도 측정
: 전체 문장에 대한 representation을 학습할 수 있도록 함

(2) Masked Decoder Self-Attention
: 각각의 출력단어가 모든 출력단어를 참고하도록 하지 않고, 본인 앞의 단어들에 한정되어 참고하도록 함

(3) Encoder-Decoder Attention
: 쿼리가 디코더에 있고, 나머지 키와 값들은 인코더에 있는 경우
: "I like you" 라는 문장이 인코더에 들어옴, 출력 문장이 "난 널 좋아해" 라고 나온다고 한다면
-> 각각의 출력 단어들이 입력 단어들과의 가중치를 측정함
: 이 과정에서 디코더의 쿼리가 인코더의 키와 값을 참조하는 방식이 적용됨

(7) 트랜스포머의 동작 원리 : Positional Encoding



Positional Encoding은 **주기 함수를 활용한 공식**을 사용
각 단어의 상대적인 위치 정보를 네트워크에 입력하게 된다
핵심은 모델이 주기를 학습하도록 하는 것 ! -> 학습이 성공적일 시 어떤 주기함수가 사용되던 성능상의 차이가 없음