Attention Is All You Need

Main Keyword

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Main Keyword

- Scaled dot product attention
- Multi head attention
- positional encoding

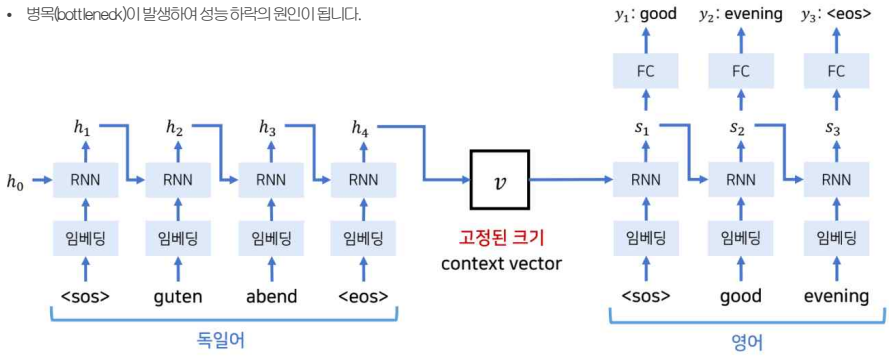
Part 1,

Background

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

context vector에 소스 문장의 정보를 압축합니다.

- 병목(bottleneck)이 발생하여 성능 하락의 원인이 됩니다.



Part 1,

Background

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

[문제 상황]

- 하나의 문맥 벡터가 소스 문장의 모든 정보를 가지고 있어야 하므로 성능이 저하됩니다.

[해결 방안]

- 그렇다면 **매번 소스 문장에서의 출력 전부를 입력으로** 받으면 어떨까요?
- 최신 GPU는 많은 메모리와 빠른 병렬 처리를 지원합니다.

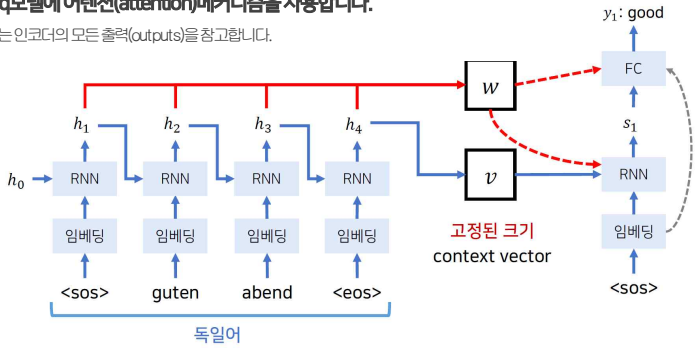
Part 1,

Seq2Seq with Attention : Decoder

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Seq2Seq모델에 어텐션(attention)메커니즘을 사용합니다.

- 디코더는 인코더의 모든 출력(outputs)을 참고합니다.



memory와 computation 때문에 embedded vector의 maximum length를 제한해야한다. 긴 sequence 데이터를 처리해야할 때, 제한된 크기의 vector로 모든 정보를 담아내야하기 때문에 정보의 손실이 커지고 이에 따라 성능의 병목현상이 일어난다.

Part 1, Background

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Sequential 문제를 풀기 위한, 이전 연구

- LSTM, GRU 등을 활용한, "Recurrent (순환) 구조"가 언어 모델링 및 기계번역 등의 Task에서 확고한 입지를 다져왔음
- Recurrent (순환) 구조는 Input과 Output Sequence를 활용
- $h_t : t-1$ 를 Input과 $h_1 - 1$ 를 통해 생성 → 이러한 구조로 인해 일괄처리가 제한됨 (순차적으로 t 이전의 Output이 다 계산되어야 최종 Output이 생성)
- 병렬화 제한됨
- Seqeunce의 길이가 길어질수록 더 취약해짐 → Factorization Trick이나 Conditional Computation으로 계산 효율성을 증대했으나 한계점 존재

Sequential Computation 문제를 풀기 위한, 이전 연구

- CNN구조를 통해 병렬화를 고려했지만, 여전히 한계점 존재
  - 인코더와 디코더를 연결하기 위한 추가 연산 필요
  - 원거리 Position 간의 Dependencies(종속성)을 학습하기 어려움(CNN을 활용한 병렬화 방안)

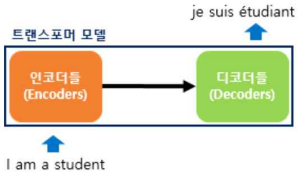
Part 2,

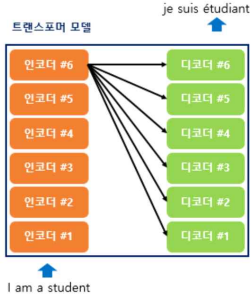
Transformer

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

RNN을 사용하지 않지만 기존의 seq2seq처럼 인코더에서 입력 시퀀스를 입력 받고, 디코더에서 출력 시퀀스를 출력하는 인코더-디코더 구조를 유지하고 있습니다.

- 이전 seq2seq구조에서는 인코더와 디코더에서 각각 하나의 RNN이 t개의 시점(time step)을 가지는 구조였다면 이번에는 인코더와 디코더라는 단위가 N개로 구성되는 구조입니다.



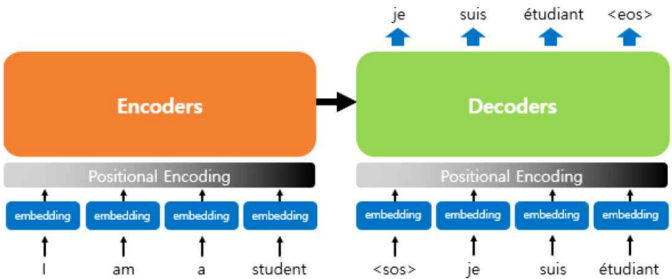


Part 2,

Transformer : Encoder와 Decoder

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **Transformer에서도 Encoder와 Decoder의 구조를 따릅니다.**
- **이때 RNN을 사용하지 않으며 인코더와 디코더를 다수 사용한다는 점이 특징입니다.**



Part 2,

Transformer : Encoder와 Decoder

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **Encoder, Decoder가 각각 N개 Stack된 Encoder-Decoder구조**
- **Decoder에서 출력되는 최종 Output은 소수로 이루어진 벡터 1개가 남게 되는데 Linear & Softmax layer를 거쳐 확률로 나타내진 벡터를 통해 문장을 만들어냄**
  - Linear layer는 fully-connected NN으로 Output 벡터를 보다 큰 Size벡터로 투영시킴
  - Size가 늘어난 벡터를 softmax를 통해 확률값으로 변환
  - 가장 높은 확률을 가진 셀의 해당하는 단어가 하나의 위치마다 하나씩 출력

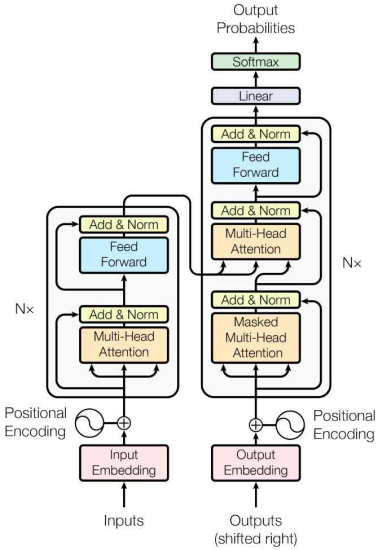


Figure 1: The Transformer - model architecture.

Part 2,

Transformer : Encoder

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **multi-head attention layer, Feed-Forward layer 과 그 둘을 연결하고 Normalization 하는 residuals 로 구성**
- **$N = 6$ 개의 동일한 Layer Stack으로 구성**
- **각 Layer는 2개의 Sub-Layer로 구성**
  - **Mulit-Head Self-Attention 메커니즘**
    - self attention 에 Multi-Head를 적용한 것
    - 같은 단어의 다른 뜻을 구분하기 위해 encoding의 target이 되는 단어 외 다른 단어들에서 힌트를 얻어 target 단어를 더 잘 encoding하기 위해 self attention을 사용
  - **Position-wise Fully Connected Feed-Forward Network**
    - 1x1 Conv Layer가 2개 이어진 것과 같은 역할
    - Position 별로 동일한 Fully Connected Feed-Forward Network가 적용

Part 2,

Transformer : Encoder

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **self attention을 사용하는 이유는 같은 단어의 다른 뜻을 구분하기 위해 encoding의 target이 되는 단어 외 다른 단어들에서 힌트를 얻어 target 단어를 더 잘 encoding 하기 위함이다.**
- **LayerNorm(x + SubLayer(x))**
  - Residual Connection 적용을 용이하게 하기 위해, Sub-layer, Embedding, Output Dimension을 512로 통일

Part 2,

Transformer : Decoder

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **encoder output을 입력을 받은 후, Attention layer에서 벡터 K, V로 변환**
- **두 벡터는 encoder-decoder attention layer에서 decoder가 input sequence의 적절한 위치에 집중하도록 돕는 역할**
- **$N = 6$ 개의 동일한 Layer Stack으로 구성**
- **각 Layer는 3개의 Sub-Layer로 구성**
  - **Masked Mulit-Head Self-Attention 메커니즘**
    - Decoder의 Mulit-Head Self-Attention은 Masking을 사용
    - Subsequent Position은 Attention 안 하도록 조정
    - Position_i를 예측할 때는 , i 보다 작은 위치에 알려진 Output에만 의존
    - Output Embedding은 One Position 씩 Offset
  - **Encoder의 Output에 대해 Mulit-Head Self-Attention 메커니즘 수행**
  - **Position-wise Fully Connected Feed-Forward Network**

Part 2,

Transformer : Encoder와 Decoder 차이점

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **Encoder는 Self-attention layers 구조로 구성**
  - **이전 Encoder Layer에서의 출력**
  - **현재 Encoder Layer에서의 동일 Position의 입력**
  - **각 Encoder Layer는 이전 Layer로부터 모든 위치를 처리한 정보를 활용**
- **Decoder는 Seq2Seq모델에서의 일반적인 Encoder-Decoder의 Attention 메커니즘을 모방**
  - **Query**
    - 이전 Decoder 하단 Layer에서 발생
  - **Key와 Value**
    - encoder의 output에서 가져
  - **각Decoder는 Encoder의 모든 Position정보를 사용함**
- **output 내 현재 위치의 이전 위치에 대해서만 attention진행**

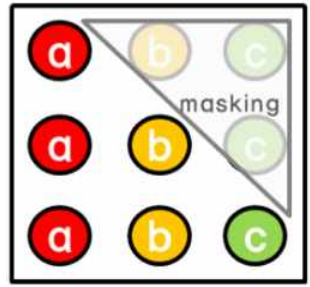
Part 2,

Transformer : Encoder와 Decoder 차이점

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **decoder에서는 encoder와 달리 순차적으로 결과를 만들어내야 하기 때문에, self-attention을 변형합니다. 바로 masking을 해주는 것이죠.** masking을 통해, position i 보다 이후에 있는 position에 attention을 주지 못하게 합니다. 즉, position i 에 대한 예측은 미리 알고 있는 output들에만 의존을 하는 것입니다.

- **예시를 보면, a를 예측할 때는 a이후에 있는 b,c에는 attention이 주어지지 않는 것입니다. 그리고 b를 예측할 때는 b이전에 있는 a만 attention이 주어질 수 있고 이후에 있는 c는 attention이 주어지지 않는 것이죠.**

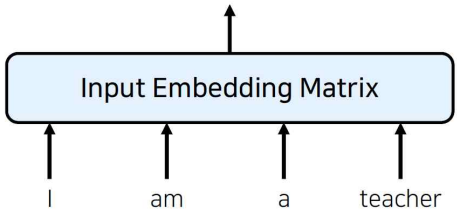


Part 3,

Transformer : 입력 값 임베딩(Embedding)

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Transformer 이전의 전통적인 임베딩은 다음과 같습니다.



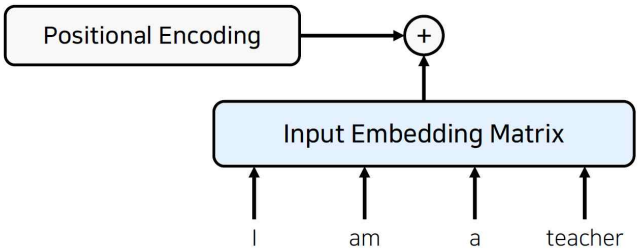
Part 3,

Transformer : 입력 값 임베딩(Embedding)

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

RNN을 사용하지 않으려면 위치 정보를 포함하고 있는 임베딩을 사용해야 합니다.

- **이를 위해 트랜스포머에서는 Postional Encoding을 사용합니다.**



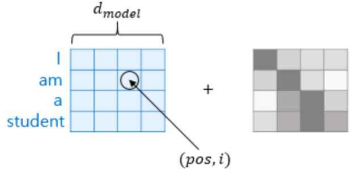
Transformer : Positional Encoding

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- Transformer 계열의 모든 방법에서 등장하는 개념
- CNN과 결정적인 차이
- Transformer는 Convolution이나 Recurrence를 사용하지 않음
- Recurrence Layer와 다르게, Multi-Head Attention Layer와 Position-wise Feed-forward Network는 Sequence와 독립적으로 계산됨

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

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



Part 3,

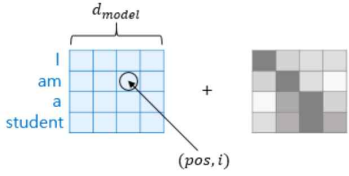
Transformer : Positional Encoding

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **Sequence와 독립적인 특징**
  - 계산을 병렬로 수행할 수 있음
  - Sequence 정보를 모델링하는데, 한계가 있을 수 있음
- **Sequence의 순서 정보를 넣어주기 위해선, 상대 또는 절대 위치 정보를 주입이 필수**
  - Transformer는 Position Encoding를 적용
- **Position Encoding을 Encoder와 Decoder의 입력 임베딩에 추가**
  - Position Encoding의 Dimension d_{model}=512
  - 입력 임베딩과의 합산을 위해서, 동일 Dimension (d_{model}=512)을 사용

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

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



Part 3,

Transformer : Positional Encoding

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

$$PE_{(pos,2i)} = sin(\frac{pos}{c})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{c})$$

$$PE_{(pos+k,2i)} = sin(\frac{pos+k}{c}) = sin(\frac{pos}{c})cos(\frac{k}{c}) + cos(\frac{pos}{c})sin(\frac{k}{c}) = PE_{(pos,2i)}cos(\frac{k}{c}) + cos(\frac{pos}{c})sin(\frac{k}{c})$$

$$PE_{(pos+k,2i+1)} = cos(\frac{pos+k}{c}) = cos(\frac{pos}{c})cos(\frac{k}{c}) - sin(\frac{pos}{c})sin(\frac{k}{c}) = PE_{(pos,2i+1)}cos(\frac{k}{c}) - sin(\frac{pos}{c})sin(\frac{k}{c})$$

- **수식 설명**
  - 홀수 : Cosine
  - 짝수 : Sine
  - i : 임베딩 차원의 위치 • pos : Word의 위치
- **Sinusoid로 구성(절대적 거리)의 장점**
  - 학습과정에서 만나지 못한 긴 문장도 대응 가능
  - 후속 연구들에서는 상대적(Relative) 거리의 Positional Encoding을 자주 사용

Part 3,

Transformer : Positional Encoding

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

$$PE_{(pos,2i)} = sin(\frac{pos}{c})$$

$$PE_{(pos,2i+1)} = cos(\frac{pos}{c})$$

$$PE_{(pos+k,2i)} = sin(\frac{pos+k}{c}) = sin(\frac{pos}{c})cos(\frac{k}{c}) + cos(\frac{pos}{c})sin(\frac{k}{c}) = PE_{(pos,2i)}cos(\frac{k}{c}) + cos(\frac{pos}{c})sin(\frac{k}{c})$$

$$PE_{(pos+k,2i+1)} = cos(\frac{pos+k}{c}) = cos(\frac{pos}{c})cos(\frac{k}{c}) - sin(\frac{pos}{c})sin(\frac{k}{c}) = PE_{(pos,2i+1)}cos(\frac{k}{c}) - sin(\frac{pos}{c})sin(\frac{k}{c})$$

- **Positional Encoding의 조건**
  - 고정 오프셋(Offset) K가 있을 때, 선형 변환을 통해 표현 가능해야함
  - 즉, PE_{pos+K}는 PE_{pos}의 선형식으로 표현 가능해야함
  - Sinusoid로 구성(절대적 거리)은 선형식으로 표현 가능
- **Attention을 활용하면 상대적인(Relative) Position을 쉽게 학습할 수 있다는 가정하에, Sinusoid로 구성(절대적 거리)**

Part 3,

Transformer : Positional Encoding

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}})$$
$$PE_{pos} =$$
$$[cos(pos/1), sin(pos/10000^{2i/d_{model}}, cos(pos/10000^{2i/d_{model}}), ..., sin(pos/10000)]$$

- **수식 설명**
  - 홀수 : Cosine
  - 짝수 : Sine
  - i : 임베딩 차원의 위치 • pos : Word의 위치
- **Sinusoid로 구성(절대적 거리)의 장점**
  - 학습과정에서 만나지 못한 긴 문장도 대응 가능
  - 후속 연구들에서는 상대적(Relative) 거리의 Positional Encoding을 자주 사용

Part 3,

Transformer : Positional Encoding

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

$$PE_{pos,2i} = sin(pos/10000^{2i/d_{model}})$$
$$PE_{pos,2i+1} = cos(pos/10000^{2i/d_{model}})$$
$$PE_{pos} =$$
$$[cos(pos/1), sin(pos/10000^{2i/d_{model}}, cos(pos/10000^{2i/d_{model}}), ..., sin(pos/10000)]$$

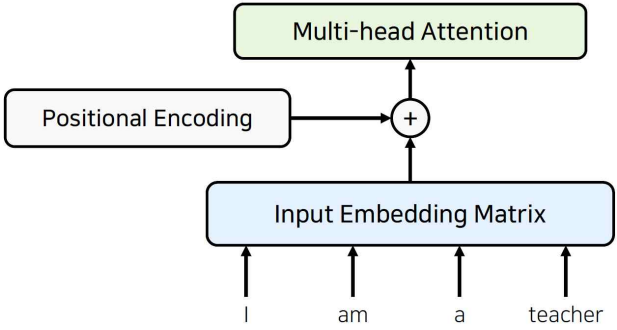
- **Positional Encoding의 조건**
  - 고정 오프셋(Offset) K가 있을 때, 선형 변환을 통해 표현 가능해야함
  - 즉, PE_{pos+K}는 PE_{pos}의 선형식으로 표현 가능해야함
  - Sinusoid로 구성(절대적 거리)은 선형식으로 표현 가능
- **Attention을 활용하면 상대적인(Relative) Position을 쉽게 학습할 수 있다는 가정하에, Sinusoid로 구성(절대적 거리)**

Part 3,

Transformer : 입력 값 임베딩(Embedding)

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **임베딩이 끝난 이후에 어텐션(Attention)을 진행합니다.**



Part 3,

Transformer : Self-Attention의 필요성

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **Recurrent 및 Convolutional Layer와의 비교**
- **세 가지 측면을 고려**
  - **Layer당 Computational Complexity**
  - **병렬 작업(Parallelzed Computation)이 필요한 양**
    - 필요한 순차 작업의 최소 수
    - Minimum Number of Sequential Operations required)

- **네트워크 내부에서 장거리 종속 성간 경로의 최대 길이**
  - **Path Length between long-range Dependencies in Network**
  - **Maximum path length between any two input and output positions in networks**
  - **장거리 종속성(dependencies)을 학습하는 것은 Sequence Transduction Task에서 핵심적인 과제**
  - **Input 및 Output Sequence에서의 Position 간의 경로가 짧을수록, 더 쉽게 장거리(Long-Range) 종속성을 학습 가능**
  - **네트워크 내 노드에서 정보를 교환하기 위해서, 통과해야 되는 경로의 길이**
  - **두 입출력 Position 사이의 최대 경로 길이도 함께 비교**

Part 3,

Transformer : Self-attention

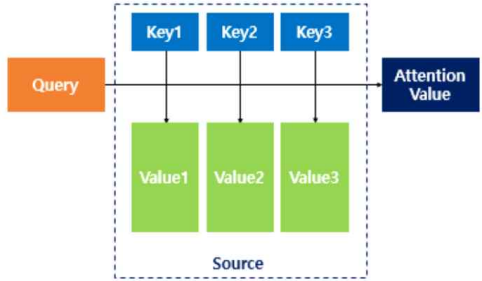
"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

인코더와 디코더는 Multi-Head Attention 레이어를 사용합니다.

- attention을 위한 세 가지 입력 요소

- **쿼리(Query) : t시점의 디코더 셀에서의 은닉 상태**
- **키(Key) : 모든 시점의 인코더 셀의 은닉 상태들**
- **값(Value) : 모든 시점의 인코더 셀의 은닉 상태들**

- **쿼리(Query) : 입력 문장의 모든 단어 벡터들**
- **키(Key) : 입력 문장의 모든 단어 벡터들**
- **값(Value) : 입력 문장의 모든 단어 벡터들**



Part 3,

Transformer : Self-attention

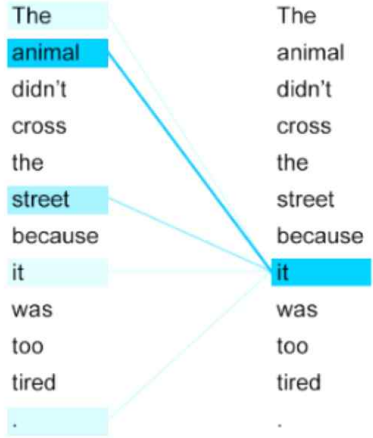
"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

인코더와 디코더는 Multi-Head Attention 레이어를 사용합니다.

- attention을 위한 세 가지 입력 요소

  - **쿼리(Query) : t시점의 디코더 셀에서의 은닉 상태**
  - **키(Key) : 모든 시점의 인코더 셀의 은닉 상태들**
  - **값(Value) : 모든 시점의 인코더 셀의 은닉 상태들**

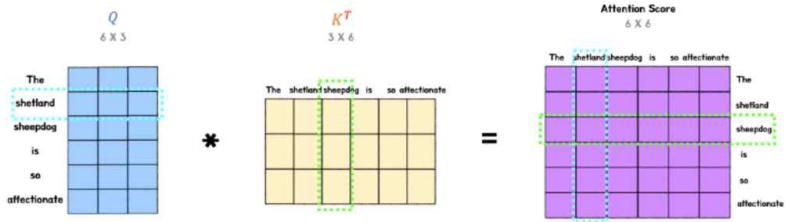
- **쿼리(Query) : 입력 문장의 모든 단어 벡터들**
- **키(Key) : 입력 문장의 모든 단어 벡터들**
- **값(Value) : 입력 문장의 모든 단어 벡터들**



Part 3,

Transformer : Attention Score

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조



query와 key 행렬의 행렬곱으로 이때 행렬곱은 행렬 간의 유사도를 의미하기에 Attention Score는 query와 key사이의 유사도인 유사도 행렬을 나타낸다.

Part 3,

Transformer : Cosine 유사도

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

$$\cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{A \cdot B}{scaling} = similarity(A, B) \qquad \text{similairty}(A, B) = \frac{A \cdot B^T}{scaling}$$

코사인 유사도는 두 벡터가 유사할 수록 값이 1에 가까워지고 두 벡터가 서로 다를 수록 -1에 가까워지는 특징을 가지고 있다. 이러한 코사인 유사도를 다음과 같이 표현할 수 있다.

즉 코사인 유사도는 벡터의 곱을 두 벡터의 L2 norm 곱, 즉 스케일링으로 나눈 값이다. 다시말해 벡터A와 벡터B의 유사도를 구하는 벡터 유사도를 뜻한다.

이를 토대로 우리는 이와 같은 행렬 유사도도 구할 수 있게 된다. 행렬A와 행렬B의 유사도로 행렬 유사도의 경우 행렬 간의 곱으로 발생하는 차원 충돌을 피하기 위해 행렬B를 전치행렬처리 해준다.

Part 3,

Transformer : Cosine 유사도

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

$$\text{similairty}(Q, K) = \frac{Q \cdot K^T}{scaling} \longrightarrow Attention(Q, K, V) = softmax\left\{\frac{Q \cdot K^T}{\sqrt{d_k}}\right\} V$$

앞서 구한 행렬 간의 유사도에서 행렬A와 행렬B 대신 우리가 구하고자 하는 행렬 query와 행렬 key를 넣어주면 앞서 살펴본 Attention Score를 구하는 식이 된다

Part 3,

Transformer : Cosine 유사도

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

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

- **Additive attention과 Dot-product attention을 일반적으로 사용**
  - **Additive attention은 Single Hidden Layer로 구성된 Feed-Forward Network를 활용 → Compatibility(일치성) 계산**
  - **Additive attention란**
    - Dot-product(Multiplicative) Attention은 효율적인 행렬곱 구성으로, 실제로 더 빠르고 공간 효율적이지만 작은 Key 벡터의 차원 d_k 에서는 두 메커니즘이 유사한 성능을 보이지만, 더 큰 d_k 에서는 Additive Attention이 더 좋음

Part 3,

Transformer : Cosine 유사도

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

$$QW_i^Q = [d_Q \times d_{model}] \times [d_{model} \times d_k] = [d_Q \times d_k]$$

$$KW_i^K = [d_K \times d_{model}] \times [d_{model} \times d_k] = [d_K \times d_k]$$

$$VW_i^V = [d_V \times d_{model}] \times [d_{model} \times d_v] = [d_V \times d_v]$$

↓

$$Attention(QW_i^Q, KW_i^K, VW_i^V) = [d_V \times d_v]$$

↓

$$Concat(QW_i^Q, KW_i^K, VW_i^V)W^O = [d_V \times hd_v] \times [hd_v \times d_{model}] = [d_V \times d_{model}]$$

- dQ,dK,dV 는 각각 query, key, value 개수

Part 3,

Transformer : 어텐션(Attention)

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

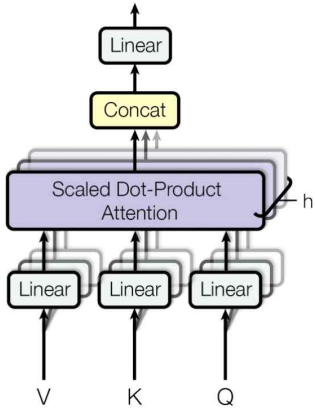
- **Incoder와 decoder는 Multi-Head Attention layer를 사용합니다.**

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

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

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

h: 헤드(head)의 개수



Multi-Head Attention

Part 3,

Transformer : 어텐션(Attention)

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

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

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

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

h: 헤드(head)의 개수

- 먼저 input은 d_k dimension의 query와 key들, dv dimension의 value들로 이루어져 있습니다.
- 이때 모든 query와 key에 대한 dot-product를 계산하고 각각을 $\sqrt{d_k}$ 로 나누어줍니다. dot-product를 하고 $\sqrt{d_k}$ 로 scaling을 해주기 때문에 Scaled Dot-Product Attention인 것입니다. 그리고 여기에 softmax를 적용해 value들에 대한 weights를 얻어냅니다.

Part 3,

Transformer : 어텐션(Attention)

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

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

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

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

h: 헤드(head)의 개수

- key와 value는 attention이 이루어지는 위치에 상관없이 같은 값을 갖게 됩니다. 이때 query와 key에 대한 dot-product를 계산하면 각각의 query와 key 사이의 유사도를 구할 수 있게 됩니다. 흔히 들어본 cosine similarity는 dot-product에서 vector의 magnitude로 나눈 것입니다. $\sqrt{d_k}$ 로 scaling을 해주는 이유는 dot-products의 값이 커질수록 softmax 함수에서 기울기의 변화가 거의 없는 부분으로 가기 때문입니다.

Part 3,

Transformer : 어텐션(Attention)

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

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

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

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

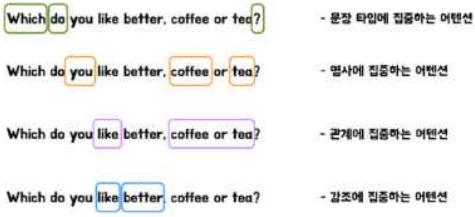
h: 헤드(head)의 개수

- **softmax를 거친 값을 value에 곱해준다면, query와 유사한 value일수록, 즉 중요한 value일수록 더 높은 값을 가지게 됩니다. 중요한 정보에 더 관심을 둔다는 attention의 원리에 알맞은 것입니다.**

Part 3,

Transformer : Multi head attention을 하는 이유

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조



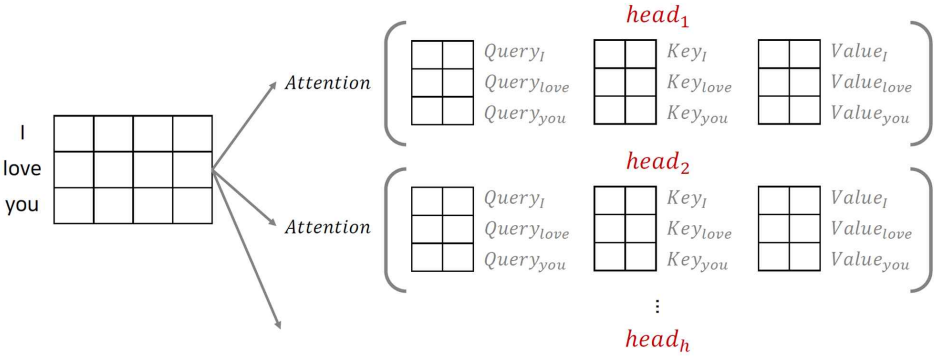
- 병렬로 multi-head를 사용함으로 여러 부분에 동시에 어텐션을 가할 수 있어서 모델이 입력 토큰 간의 다양한 유형의 종속성을 포착하고 동시에 모델이 다양한 소스의 정보를 결합할 수 있게 된다.
- 위 그림과 같이 한 head는 문장 타입에 집중하는 어텐션을 줄 수도 있고, 다른 head는 명사에 집중하는 어텐션, 또 다른 head는 관계에 집중하는 어텐션 등등 multi-head는 같은 문장 내 여러 관계 또는 다양한 소스 정보를 나타내는 정보들에 집중하는 어텐션을 줄 수 있다.

Part 3,

Transformer : Multi-Head Attention

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

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



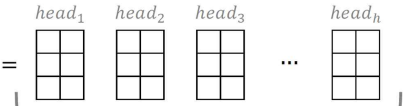
- Query, Key, Value를 서로 다른 선형 Projection 하는 것이 유리
- Scaled Dot-Product를 통해 산출되는 벡터가 여러개라 vector size가 맞지 않아 바로 전달불가

Transformer : Multi-Head Attention

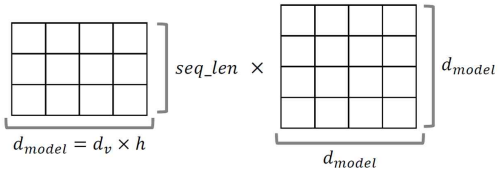
"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- 병렬 어텐션을 모두 수행하였다면 모든 어텐션 헤드를 연결(concatenate)합니다. 모두 연결된 어텐션 헤드 행렬 크기는 (seq_len, dmodel)가 됩니다. -> 차원(dimension)이 동일하게 유지

$$Concat(head_1, \dots, head_h) =$$



$$MultiHead(Q, K, V) =$$



- Head의 축소된 차원(d_k×Head=64×8=512)으로 인해, 총 계산 비용은 Single-Head Attention(d_{model}=512)과 유사
- 모든 산출 벡터를 concat한 후 1개가 된 벡터를 size가 맞는 weight matrix로 내적해 Feed-forward에 맞는 size의 벡터로 변환

Part 3,

Transformer : Multi-Head Attention 내용 요약

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

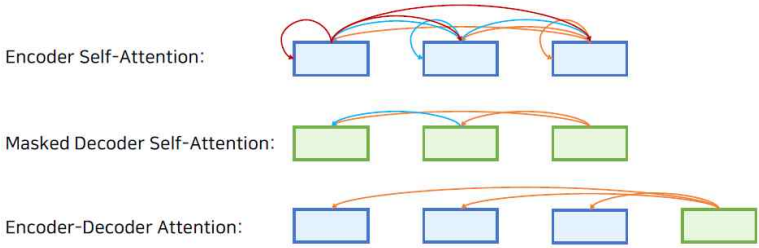
- **Query, Key, Value값을 한 번에 계산하지 않고 head 수만큼 나눠 계산 후 나중에 Attention Value들을 합치는 메커니즘. 한마디로 분할 계산 후 합산하는 방식.**
  - 원래 Query, Key, Value 행렬 값을 head 수만큼 분할
  - 분할된 행렬 값을 통해, 각 Attention value값들을 도출
  - 도출된 Attention value값들을 concatenate(쌓아 합치기)하여 최종 Attention value도출

Part 3,

Transformer : 어텐션(Attention)의 종류

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **Transformer에서는 세가지 종류의 어텐션(attention)레이어가 사용됩니다.**



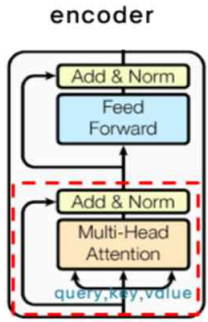
- Incoder의 self-attention: Query = Key = Value
- decoder의 masked self-attention : Query = Key = Value
- decoder의 Incoder-decoder attention = Query : decoder vector / Key = Value :Incoder vector

Part 3,

Transformer : Encoder self-attention layer

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- key, value, query들은 모두 encoder의 이전 layer의 output에서 옵니다. 따라서 이전 layer의 모든 position에 attention을 줄 수 있습니다. 만약 첫번째 layer라면 positional encoding이 더해진 input embedding이 됩니다

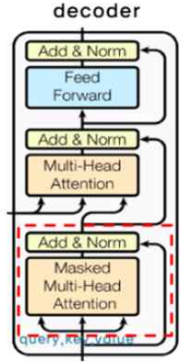


Part 3,

Transformer : Decoder self-attention layer

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- encoder와 비슷하게 decoder에서도 self-attention을 줄 수 있습니다. 하지만 i번째 output을 다시 i+1번째 input으로 사용하는 auto-regressive한 특성을 유지하기 위해 , masking out된 scaled dot-product attention을 적용했습니다.
- masking out이 됐다는 것은 i번째 position에 대한 attention을 얻을 때, i번째 이후에 있는 모든 position은 $\text{Attention}(Q,K,V)=\text{softmax}(QK^T/\sqrt{d_k})V$에서 softmax의 input 값을 $-\infty$로 설정한 것입니다. 이렇게 한다면, i번째 이후에 있는 position에 attention을 주는 경우가 없겠죠.

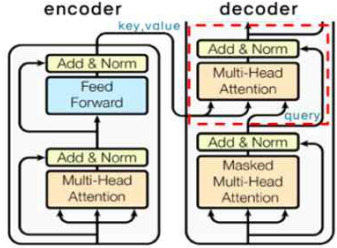


Part 3,

Transformer : Decoder self-attention layer

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- query들은 이전 decoder layer에서 오고 key와 value들은 encoder의 output에서 오게 됩니다. 그래서 decoder의 모든 position에서 input sequence 즉, encoder output의 모든 position에 attention을 줄 수 있게 됩니다.
- query가 decoder layer의 output인 이유는 query라는 것이 조건에 해당하기 때문입니다. 좀 더 풀어서 설명하면, '지금 decoder에서 이런 값이 나왔는데 무엇이 output이 돼야 할까?'가 query인 것이죠.
- 이때 query는 이미 이전 layer에서 masking out됐으므로, 1번째 position까지만 attention을 얻게 됩니다. 이 같은 과정은 sequence-to-sequence의 전형적인 encoder-decoder mechanisms를 따라한 것입니다.



Part 3,

Transformer : Position-wise Feed-Forward Networks

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

$$\mathrm{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- **input과 output의 차원은 512, inner-layer의 차원은 2048

- **Encoder 및 Decoder 모두 Position-wise Feed-Forward Networks로 구성됨**
- **Position-wise Feed-Forward Networks는 각 Position별로 적용 및 동일하게 적용**
- **Linear Transformation → ReLU → Linear Transformation**

Part 3,

Transformer : Embedding and Softmax

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

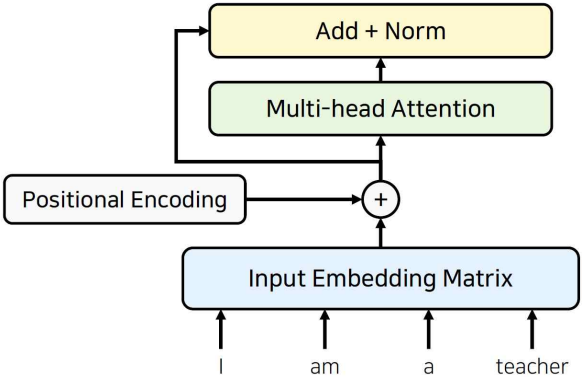
- **각 Token은 Embedding 벡터로 변환(Input, Output Embedding 및 역 Embedding 포함하여 3번)**
  - Embedding은 동일 Matrix를 사용
  - 디코더에서의 번역 결과는, Linear Transformation → Softmax → Token화 과정을 거침

Part 3,

Transformer : 입력 값 임베딩(Embedding)

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **임베딩이 끝난 이후에 어텐션(Attention)을 진행합니다.**



Part 3,

Transformer : Model summary

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

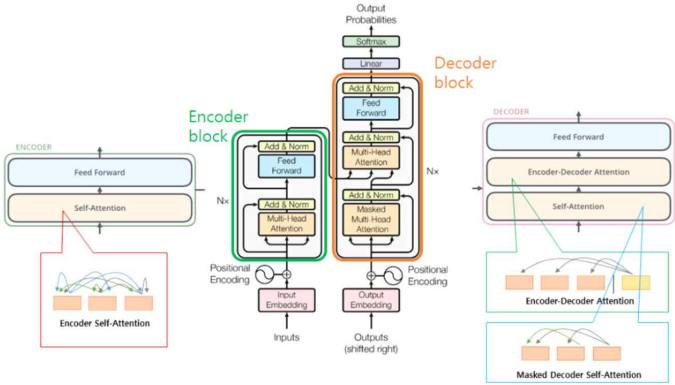


그림13. Transformer 모델 전체 개요.

Part 3,

Transformer : Self-Attention의 필요성 결과

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- 'n'은 시퀀스 길이
- 'd'는 표현 차원
- 'k'는 컨볼루션의 커널 크기
- 'r'은 제한된 자기주의(self-attention)에서의 이웃의 크기

Part 3,

Transformer : Self-Attention의 필요성 결과

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

Complexity per Layer

- n < d 인 경우 Self-attention이 Recurrent 보다 빠름
- 기계 번역 SOTA 모델에서 자주 발생되는 상황
- 매우 긴 Sequence인 경우, Computational Performace 향상을 위해, Self-Attention은 이웃(Neighborhood) r만큼으로 제한할 수 있음
- 단 Maximum Path Length는 O(n/r) 증가
- 복잡성은 'Big O' 표기법으로 표현되며, 이는 최악의 경우에 대한 성장 속도를 나타냄
- 복잡성이 높을수록 계층을 계산하는 데 더 많은 시간과 컴퓨팅 자원이 필요

Part 3,

Transformer : Self-Attention의 필요성 결과

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- **Sequential Operations**
  - 계산을 수행하는 동안 필요한 순차적 단계의 수
  - 순차적 연산의 수가 많을수록 병렬 처리가 어렵고, 따라서 계산 속도가 느려질 수 있음
  - O(1)은 계산이 순차적이지 않고 병렬로 수행될 수 있음을 의미
  - O(n)은 시퀀스의 각 요소를 순차적으로 처리해야함을 의미

Part 3,

Transformer : Self-Attention의 필요성 결과

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- **Self-Attention**
  - 입력 시퀀스의 각 요소가 서로 얼마나 관련이 있는지를 계산
  - 모델이 입력 데이터에서 중요한 부분에 주목
  - 복잡성은O(n^2 · d)로, 시퀀스 길이와 표현 차원에 제곱비례
  - 순차적 연산은O(1)로, 병렬 처리가 가능하기 때문에 연산의 수가 적음
  - 최대 경로 길이도 O(1)로, 어떤 입력도 직접 다른 입력에 연결될 수 있음을 의미

Part 3,

Transformer : Self-Attention의 필요성 결과

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- **Recurrent**
  - RNN은 이전에 계산된 출력을 현재 입력과 결합하여 시퀀스 데이터를 처리
  - 복잡성은 $O(n^2 \cdot d)$ 로, 퀀스 길이와 표현 차원에 선형 비례
  - 순차적 연산은 $O(n)$ 로, 각 시간 단계마다 연산이 필요하므로 순차적으로 연산량 증가
  - 최대 경로 길이도 $O(n)$ 로, 시간에 따라 정보 전달

Part 3,

Transformer : Self-Attention의 필요성 결과

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- **Convolutional**
  - CNN은 주로 이미지 처리에 사용
  - 복잡성은 $O(k \cdot n \cdot d^2)$ 로, 커널 크기에 선형적으로 비례
  - 순차적 연산은 $O(1)$ 로, 병렬 처리가 가능
  - 최대 경로 길이는 $O(\log(n))$ 로, 계층적 구조 덕분에 로그 시간 안에 정보 전파

Part 3,

Transformer : Self-Attention의 필요성 결과

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 1: Maximum path lengths, per-layer complexity and minimum number of sequential operations for different layer types. n is the sequence length, d is the representation dimension, k is the kernel size of convolutions and r the size of the neighborhood in restricted self-attention.

Layer Type	Complexity per Layer	Sequential Operations	Maximum Path Length
Self-Attention	$O(n^2 \cdot d)$	$O(1)$	$O(1)$
Recurrent	$O(n \cdot d^2)$	$O(n)$	$O(n)$
Convolutional	$O(k \cdot n \cdot d^2)$	$O(1)$	$O(log_k(n))$
Self-Attention (restricted)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

- **Self-Attention(restricted)**
  - 일반 자기 주의 메커니즘의 변형
  - 산 복잡성을 줄이기 위해 이웃의 크기를 제한
  - 복잡성은 $O(r \cdot n \cdot d)$ 로, 제한된 이웃의 크기에 선형적으로 비례
  - 순차적 연산은 $O(1)$
  - 최대 경로 길이는 $O(n/r)$

Part 4,

Transformer : Training

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **Training Data 및 Batch**
  - Training Data 약 450 만 문장 쌍(영어-독일어 세트) – 공유된 약 37000 Token 사용
  - Training Data 약 3600만 문장 쌍(영어-프랑스어 세트) – 공유된 약 32000 Token 사용
  - Byte Pair Encoding(BPE) 사용

- **Regularization**
  - **Dropout : $P_{drop}=0.1$**
    - 각 Sub-Layer Output(Residual Connection 및 Normalization) 전에 Dropout 적용
    - Encoder 및 Decoder에 적용
      - "Embedding" 및 "Positional Encoding" 합 이후에 Dropout 적용
  - **Label Smoothing : $varepsilon_{ls}=0.1$**
    - 불확실함을 추가 학습하여, 정확도와 BLEU 점수 향상

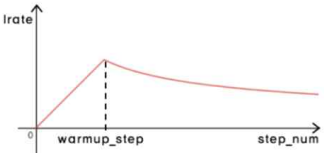
Part 4,

Transformer : Training

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

We used the Adam optimizer [20] with $\beta_1 = 0.9$, $\beta_2 = 0.98$ and $\epsilon = 10^{-9}$. We varied the learning rate over the course of training, according to the formula:

$$lrate = d_{\text{model}}^{-0.5} \cdot \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$



- **Optimizer**
  - 많이 쓰이는 Adam optimizer를 사용했습니다.
  - 특이한 점은 learning rate를 training동안 고정시키지 않고 다음 식에 따라 변화시켰다는 것입니다.
  - warmup_step 까지는 linear하게 learning rate를 증가시키다가, warmup_step 이후에는 step_num의 inverse square root에 비례하도록 감소시킵니다
  - 이렇게 하는 이유는 처음에는 학습이 잘 되지 않은 상태이므로 learning rate를 빠르게 증가시켜 변화를 크게 주다가, 학습이 꽤 됐을 시점에 learning rate를 천천히 감소시켜 변화를 작게 주기 위해서입니다

Part 4,

Transformer : Training

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- **상대적으로 적은 Training Cost로 우수한 성능, 심지어 Ensemble 모델보다 우수**
  - Base Model : 5 Checkpoint의 평균으로 얻어진 Single Model을 사용
  - Big Model : 20 Checkpoint의 평균으로 얻어진 Single Model을 사용

Transformer : Training

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

Table 2: The Transformer achieves better BLEU scores than previous state-of-the-art models on the English-to-German and English-to-French newstest2014 tests at a fraction of the training cost.

Model	BLEU		Training Cost (FLOPs)	
	EN-DE	EN-FR	EN-DE	EN-FR
ByteNet [18]	23.75			
Deep-Att + PosUnk [39]		39.2		$1.0 \cdot 10^{20}$
GNMT + RL [38]	24.6	39.92	$2.3 \cdot 10^{19}$	$1.4 \cdot 10^{20}$
ConvS2S [9]	25.16	40.46	$9.6 \cdot 10^{18}$	$1.5 \cdot 10^{20}$
MoE [32]	26.03	40.56	$2.0 \cdot 10^{19}$	$1.2 \cdot 10^{20}$
Deep-Att + PosUnk Ensemble [39]		40.4		$8.0 \cdot 10^{20}$
GNMT + RL Ensemble [38]	26.30	41.16	$1.8 \cdot 10^{20}$	$1.1 \cdot 10^{21}$
ConvS2S Ensemble [9]	26.36	**41.29**	$7.7 \cdot 10^{19}$	$1.2 \cdot 10^{21}$
Transformer (base model)	27.3	38.1	**$3.3 \cdot 10^{18}$**	
Transformer (big)	**28.4**	**41.8**	$2.3 \cdot 10^{19}$	

- **Beam Search 사용 : Beam Size 4, Length Penalty alpha = 0.6**
  - 학습이 완료 후, Dev Set에서 Beam Search와 관련된 최적의 Hyper-Paramter를 선택
  - 번역 시, 최대 출력 길이 : 입력 길이 + 50 (가능하면 짧게 번역)
  - **Beam Search**
    - Machine Translation에서 사용되는 기술
    - 각 Step에서 Beam Size 만큼을 함께 고려하면서, 최적의 조건부 확률을 가지는 문맥을 선정해 나아가는 방법
    - Beam Size를 크게 하면, 번역 선능은 높아지나, 디코딩 속도가 저하

Part 4,

Transformer : Training

"Transformer" : 온전히 attnetion mechanism에만 기반한 구조

- **Model Averaging 사용하지 않음**
- **Beam Search 사용**
- **(A) Multi-Head**
  - Single Head이나 너무 많은 Head는 성능이 저하됨
- **(B) Attention의 Dimension**
  - d_k를 줄이면 품질이 저하됨
  - Dot-Product Attention 보다 더 정교한 "Compatibility 함수"가 있을 수 있음을 시사
- **(C) Model의 크기**
  - 더 큰 모델이 더 좋은 성능
- **(D) 과적합**
  - 과적합 방지가 성능에 더 좋음
  - Dropout은 Overfitting을 완화해줌
- **(E) Learned Positional Embedding**
  - 성능이 거의 동일
  - 고정된 Sinusoids 써도 됨

Table 3: Variations on the Transformer architecture. Unlisted values are identical to those of the base model. All metrics are on the English-to-German translation development set, newstest2013. Listed perplexities are per-wordpiece, according to our byte-pair encoding, and should not be compared to per-word perplexities.

	N	d_{model}	d_{ff}	h	d_k	d_v	P_{drop}	ϵ_{ls}	train steps	PPL (dev)	BLEU (dev)	params $\times 10^6$
base	6	512	2048	8	64	64	0.1	0.1	100K	4.92	25.8	65
(A)				1	512	512				5.29	24.9	
				4	128	128				5.00	25.5	
				16	32	32				4.91	25.8	
				32	16	16				5.01	25.4	
(B)					16					5.16	25.1	58
					32					5.01	25.4	60
(C)	2									6.11	23.7	36
	4									5.19	25.3	50
	8									4.88	25.5	80
		256			32	32				5.75	24.5	28
		1024			128	128				4.66	26.0	168
			1024							5.12	25.4	53
			4096							4.75	26.2	90
(D)							0.0			5.77	24.6	
							0.2			4.95	25.5	
								0.0		4.67	25.3	
								0.2		5.47	25.7	
(E)	positional embedding instead of sinusoids									4.92	25.7	
big	6	1024	4096	16			0.3		300K	**4.33**	**26.4**	213

Q&A


감사합니다