

type 연구

자연어처리 Transformer

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin



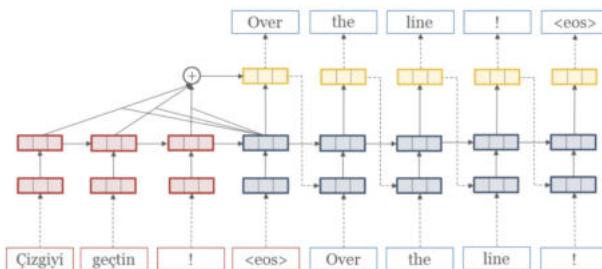
RESEARCH

OVERVIEW

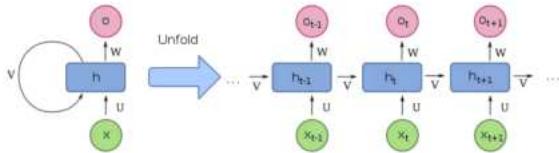
지배적인 시퀀스 변환 모델은 인코더와 디코더를 포함하는 복잡한 반복 또는 컨볼루션 신경망을 기반으로 합니다. 가장 성능이 좋은 모델도 **attention** 메커니즘을 통째로 인코더와 디코더를 연결합니다. 저희는 **attention** 메커니즘을 기반으로 하는 새로운 간단한 네트워크 아키텍처인 **transformer**를 제안하며, **recurrence**와 **convolution**을 완전히 폐지합니다. 두 가지 기계 번역 작업에 대한 실험 결과, 이 모델은 품질이 우수하면서도 병렬화가 가능하고 훈련에 훨씬 적은 시간이 소요되는 것으로 나타났습니다. 이 모델은 WMT 2014 영어-독일어 번역 작업에서 28.4 BLEU를 달성하여 양상률을 포함한 기준 쪼고 결과보다 2 BLEU 이상 개선되었습니다. WMT 2014 영어-프랑스어 번역 과제에서 우리 모델은 8개의 GPU에서 3.5일간 훈련한 후 41.8의 새로운 단일 모델 최첨단 BLEU 점수를 기록했는데, 이는 문헌에 나온 최고 모델 훈련 비용의 일부에 불과한 수준입니다. 대규모 훈련 데이터와 재현된 훈련 데이터 모두에서 영어-터키구 구문 분석에 성공적으로 적용함으로써 트랜스포머가 다른 작업에도 잘 일반화됨을 보여줍니다.

CONTENT

1. Introduction



- Sequence modeling은 어떠한 Sequence를 가지는 데이터로부터 또 다른 Sequence를 가지는 데이터를 생성하는 task이다.



- RNN, LSTM, GRU가 주축으로 사용되었는데 몇 가지 문제가 발생하였다.
- 위와 같은 Recurrent model들은 모든 데이터를 한 번에 처리하는 것이 아니라 sequence position t 에 따라 순차적으로 입력에 넣어주어야 한다.
- 이미 한 번에는 긴 sequence 길이를 가지는 데이터를 처리할 때 memory와 computation에서 많은 부담이 생기게 된다.
- 따라서 병렬적 처리를 어렵게 만든다.
- Attention mechanism은 입력과 출력 사이의 거리와 관계없이 의존성을 모델링할 수 있으나 거의 대부분의 경우 순차 네트워크와 함께 사용되고 있어 효율적인 병렬화 불가능하다.(순환 네트워크의 문제점)
- 따라서 이 논문에서 순환없이 입력값과 출력값 간 전역 의존성을 모델링할 수 있는 Attention mechanism만을 사용한 모델 구조인 Transformer 제안

2. Background

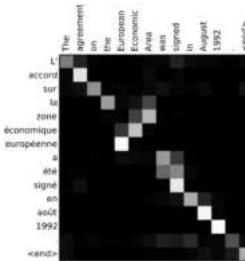
- Extended Neural GPU, ByteNet, ConvS2S에서도 연속적 연산을 줄이기 위한 연구가 이루어졌는데, 모두 CNN을 기본 구성 요소로 사용한다.
- 이러한 모델들은 입력과 출력 사이의 거리에 따라 (선형 or 로그 비례) 계산량이 증가한다.
- 따라서 입력값과 출력 값의 거리가 멀수록 의존성을 알기 매우 어렵다.

1) Self-attention

- 자신에게 주어지는 attention 기법으로 단일 sequence 안에서 서로 다른 위치에 있는 요소들의 의존성을 찾아낸다.
- 예컨대 봉해, 추상적 요약, 텍스트 포함, 학습 과정, 특별적인 문장 표현을 포함한 다양한 task에서 성공적으로 사용됨

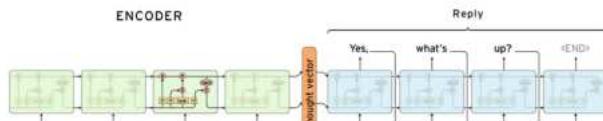
2) End-to-End memory network

- recurrent attention mechanism에 기반
- 간단한 언어 질문 답변 및 언어 모델링 작업에서 좋은 성능을 보임



- Attention은 sequence distance와 무관하게 서로 간의 dependencies를 모델링 한다.
- 예를 들어, 위 이미지는 french를 english로 번역하는 sequence modeling에서 attention을 사용할 때 그 correlation matrix를 나타낸다.
- Transformer는 이러한 Attention mechanism을 전적으로 사용하여 CNN을 사용하지 않고 오로지 Attention만을 이용해 구축하였다.

3. Model Architecture



Are you free tomorrow? <START>

Incoming Email

DECODER

- Seq2Seq 모델은 Encoder와 Decoder를 사용하는 모델이다.
- hidden state는 그 전 sequence의 정보를蓄存하고 있다.
- Encoder의 최종 output에 embedded vector를 사용하여 Decoder에 넣어주는데 **memory**와 **computation** 때문에 **embedded vector**의 maximum length를 제한하여야 한다.
- 제한된 크기의 vector로 모든 정보를 담아내야 하기 때문에 정보의 순서가 끊기고 이에 따라 성능의 별차이가 일어난다.

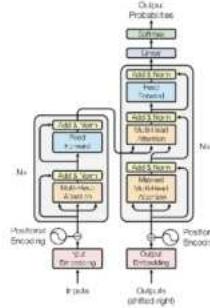


Figure 1: The Transformer - model architecture.

- 내부에는 self-attention과 fully connected layer로만 구성되어 있다.

3.1 Encoder and Decoder Stacks

Encoder:

- 6개의 동일한 layer로 이루어져 있고 각각의 layer는 2개의 sub-layer를 가진다.
- 1개는 **multi-head self-attention layer**이고 다른 1개는 **fully connected feed-forward network**로 이루어져 있다.
- 각각의 sub-layer에 대해서 **residual connection**을 이용하고 그 후 **layer normalization**을 할 것이다.
- 각각의 sub-layer의 output은 $\text{LayerNorm}(x + \text{Sublayer}(x))$

Decoder:

- 6개의 동일한 layer로 이루어져 있고 각각의 layer는 2개의 sub-layer 외에 하나의 layer를 더 가짐.

- Masked Multi-Head Decoder Self-Attention
- Encoder에서 출력으로 입력된 Multi-Head Encoder-Decoder Attention
- Feed-Forward

3.2 Attention

- Attention이란 같은 문장 내에서 단어들 간의 관계를 나타내며, 이 관계를 Query(Q), Key(K), Value(V) 3가지로 표현하여 하나의 출력 결과를 내는 것이다.

영어: I love her.

한국어: 나는 그녀를 사랑한다.

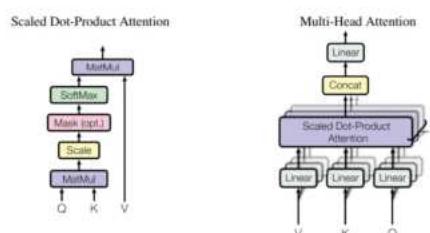
- 나는: Query, 연관성을 찾는 대상인(I, love, her)이 Key, Q와 K의 유사성을 계산하여 유사한 만큼의 V(Value)를 가지게 될 것이다.
- Q, K, V 모두 벡터이고 출력도 벡터이다.

Q(Query): 영향을 주는 벡터

K(Key): 영향을 받는 벡터

V(Value): 주는 영향의 가중치 벡터

3.2.1 Scaled Dot-Product Attention



- 위 그림은 1개의 Attention을 표현한 것이고 이러한 Attention이 여러 개 모여서 만들어진 것이 Multi Head Attention이다.

- Q, K, V 로 값을 구하는 수식은 아래와 같다.

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

- Output = value의 가중합

가중합에 이용되는 가중치: query와 연관된 key의 흐트를 향수에 의해 계산

- d_k ($k = \text{head}$ 의 개수)

<계산 과정 요약>

1. 워드 임베딩에 가중치를 곱해서 Query, Key, Value 계산

2. Query * Key = attention score → 값이 높을수록 연관성이 높고, 낮을수록 연관성이 낮다.

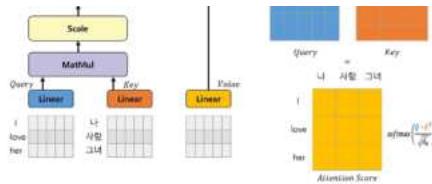
3. key의 차원수로 나누고 softmax 적용 → softmax 결과 값은 key값에 해당하는 단어가 현재 단어에 어느정도 연관성이 있는지 나타낸다.

4. 문장 속에서 지닌 일부 워드의 값 = softmax 값과 value 값을 곱하여 더한다.

→ 문장에서 차원성이 반영된 값이기 때문에 문맥을 알 수 있음.

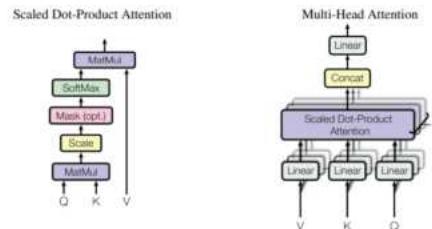
• d_k 로 나누는 이유는 d_k 값이 너무 커지게 되면 행렬의 연산의 값도 커지면서 softmax할수가 극도로 작은 기울기를 갖는 영역을 가질 것으로 추측되어 모두 0에 가까운 값으로 정규화 시켜줄려고 하는 것이다.



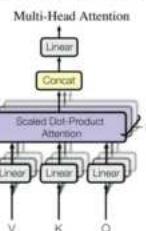


- Input Embedding의 차원이 5, 각 Head의 차원이 3이라고 가정해보자.
- Query와 Key값을 이용하여 attention score를 구해준다.
- 그 후 d_k 의 뿐만 아니라 다음 softmax를 위해 확률값을 구해준다.
- 마지막으로 Softmax로 구한 확률값에 V값을 곱하여 Attention 과정을 마무리 한다.
- Transformer는 이러한 Attention 구조가 여러개인 Multi-head Attention 구조로 이루어져 있다.

3.2.2 Multi-Head Attention



- 하나의 attention function을 사용하는 것보다 queries, keys, value는 linear projection을 통해 중간 mapping을 해줘서 각각 다른 값을 입력으로 하는 여러 개의 attention function들을 만드는 것이 더 효율적이다.
- Concat으로 여러 개의 output들을 합쳐주고 linear을 통과시킨다.
- 논문에서는 8개의 Head를 두어 Head 개수만큼 Scaled dot product Attention 연산을 수행할 수 있게 하여 모델이 다양한 관점의 Attention map을 만들도록 함.



- Input Embedding + Positional Encoding의 차원이 d라고 가정해보자.
- Attention 구조를 1번 실행시켜도 되지만 h개로 나누어 각각의 차원이 d/h 의 차원으로 조각 봉투연산하여 h개를 학습하고 다시 Concat하고 Linear Layer에 넣어주는 구조이다.

$$\begin{aligned} \text{MultiHead}(Q, K, V) &= \text{Concat}(\text{head}_1, \dots, \text{head}_h)W^O \\ \text{where } \text{head}_i &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \\ W_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_h}, W_i^K \in \mathbb{R}^{d_{\text{model}} \times d_h}, W_i^V \in \mathbb{R}^{d_{\text{model}} \times d_h}, W^O \in \mathbb{R}^{d_{\text{model}} \times d_{\text{model}}} \end{aligned}$$

- 여러 개로 나눈 Multi-Head들이 서로 다른 Representation Subspace를 학습하여 다양한 유형의 dependences을 가지 다양한 정보를 결합할 수 있다.

Which do you like better, coffee or tea?	- 문장 타입에 집중하는 어텐션
Which do you like better, coffee or tea?	- 영사에 집중하는 어텐션
Which do you like better, coffee or tea?	- 관계에 집중하는 어텐션

- 각각의 head들은 서로 다른 attention을 가지는 방식으로 동작한다.
- 그리고 원래 d 차원을 h개로 조개 별별 연산을 했기 때문에 총 계산 비용은 하나의 head를 사용했을 때와 동일하다.

3.2.3 Applications of Attention in our model

Encoder-Decoder Attention:

- Encoder의 마지막 Layer에서 출력된 Query, Key, Value는 Key, Value만 사용하고 이진 Decoder에서 Query를 사용한다.
- 이렇게 하면 Decoder의 Sequence들이 Encoder의 Sequence들과 어떠한 연관을 가지는지 학습한다.
- Decoder의 모든 위치에서 Input Sequence의 모든 위치를 참조할 수 있도록 함.

Encoder Self Attention:

- Encoder에서 사용되는 Self-Attention은 Query, Key, Value 모두 Encoder로부터 가져온다.
- 각 Layer들은 이전 layer의 output을 입력으로 받아 이전 Layer에 있는 모든 Position에 내용이 가능하고, 같은 Place의 Query마다 같은 Place의 Key, Value를로 Attention값을 구한다.
- 예시를 들면 I love her이라는 문장에서 I가 Query일 때 각 I, love, her이 Key로 내용되고 마찬가지로 love, her이 Query일 때 각각 Attention값을 구한다.

→ Encoder의 각 위치들은 이전 토큰의 모든 위치들을 참조할 수 있음.

Masked Decoder Self-Attention:

- 전반적인 목표는 Encoder와 같지만 Encoder는 입력이 Decoder는 출력 같이 들어온다.
- Decoder에서는 Sequence model의 Auto-Regressive(자기 회귀) property 보존을 위해 이후에 나올 단어들은 참조하지 않는다.
- 예시를 들면 Q-I, K-I, Q-love, K-[I, love], Q-you, K-[I, love, you]
- 이후 단어들을 참조하는 것은 철답을 알려주는 것과 같으므로 일종의 치명이다.
- 학습하고 싶지 않은 도급의 경우에는 -m로 설정하여 masking을 한다.

→ Decoder의 각 위치들은 Decoder 내의 다른 위치들을 참조할 수 있는데, 아침부터 자신 위치까지만 참조할 수 있음.

3.3 Position-wise Feed-Forward Networks

- Attention layer의 뒤에 fully connected feed-forward network가 사용됨.
- Linear → ReLU → Linear로 구성되어 있음.

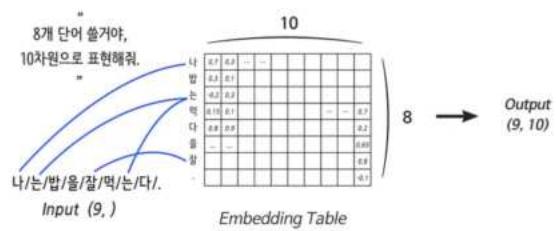
$$FFN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$

- input과 output의 차원은 512, Inner-layer의 차원은 2048

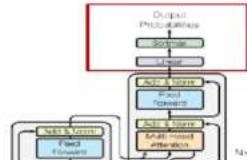
3.4 Embeddings and Softmax

Embedding 레이어

간단하게 말하면 컴퓨터를 단어 사전



- Embedding size = 단어 개수 * 단어 깊이



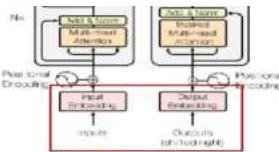


Figure 1: The Transformer - model architecture.

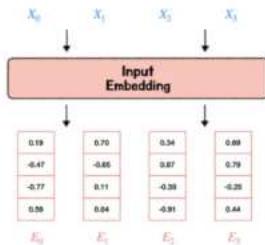
- Input Embedding \otimes Output Embedding \otimes Embedding layer을 거치고, 차원은 $d_{\text{model}} = 512$ 로 사용
- weights의 $\sqrt{d_{\text{model}}}$ 를 곱한다.
- Decoder 단 출력에서 Linear transformation을 거치고 softmax function을 하여 다음 토큰들의 확률값들을 계산한다.
- 2개의 embedding은 Linear transformation으로 동일한 weight matrix를 가진다.(공유 한다)

3.5 Positional Encoding

- 토근의 상대적인 위치(또는 절대적인 위치)에 대한 정보를 제공하기 위한 역할
- Transformer는 Recurrent model, CNN을 사용하지 않고 오직 Attention mechanism만을 사용하여 만들기 때문에 Sequence 정보를 담아낼 수 없음.
- Positional encoding을 통해 sequence 정보를 테이터에 추가해준다.

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

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



3.5.1 단어의 위치 정보가 중요한 이유

- Input embedding을 통해 각각의 단어를 embedding vector로 바꿔주었다.

① Although I did not get 95 in last TOEFL, I could get in the Ph.D program.

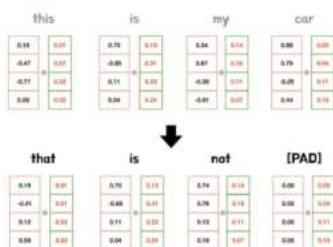
② Although I did get 95 in last TOEFL, I could not get in the Ph.D program.

- 1번과 2번 문장을 보면 Not의 위치에 따라 두 문장의 뜻이 완전히 달라지는 문제가 발생할 수 밖에 없다.

E_0	p_0	E_1	p_1	E_2	p_2	E_3	p_3
0.19		0.70		0.34		0.89	
-0.47		-0.66		0.87		0.79	
-0.77		0.11		-0.38		-0.28	
0.59		0.04		-0.91		0.44	

- 따라서 각각의 단어 벡터에 Positional Encoding을 통해 얻은 위치 정보를 더해줘야 한다.

- 여기서 반드시 지켜야 할 규칙 2가지가 있다.



1. 모든 위치값은 sequence의 길이나 input에 관계 없이 동일한 식별자를 가져야 한다.

- 따라서 sequence가 변경 되더라도 위치 임베딩은 동일하게 유지될 수 있다.

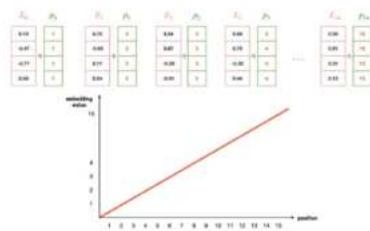




2. 모든 위치값은 너무 크면 안된다.

- 위치값이 너무 커져버리면, 단어 간의 상관관계 및 의미를 유효하게 활용할 수 있는 의미정보 값이 상대적으로 작아지게 되고 Attention layer에서 제대로 학습 및 훈련이 되지 않을 수 있다.

3.5.2 위치 벡터를 얻는 두 가지 방법과 문제점



- 첫 번째 토큰에는 0, 두 번째 토큰에는 1, ..., 둘째 sequence 크기에 미래해서 일정하게 커지는 정수값을 부여할 수 있다.

- 하지만 위치 정보 값이 급격하게 커지면 단어의 의미가 혼란될 수 있다.



- 첫 번째 토큰에는 0, 마지막 토큰은 1을 부여하고, 그 사이를 (1/단어수)로 나누어 나온 값을 normalization을 적용해볼 수 있다.

- 하지만 이 경우 같은 sequence 길이에 따라서 벡터값이 달라지기 때문에 안된다.

3.5.3 Positional Encoding을 위한 Sine & Cosine 함수

1. 의미정보가 변질되지 않도록 위치 벡터값이 너무 크면 안된다.

- sine, cosine 함수는 -1~1 사이를 반복하는 주기함수이다.

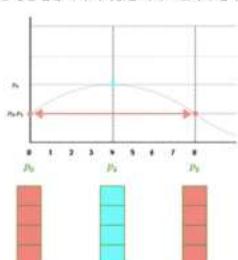
- 값이 너무 커지지 않는 조건을 만족한다.

2. 일정 구간 내에 있는 토큰들은 sine, cosine 함수를 선택한 이유는 주기함수이기 때문이다.

- sigmoid의 경우 문장의 sequence의 경우 위치 벡터값의 차이가 미미해지는 문제가 발생할 수 있다.

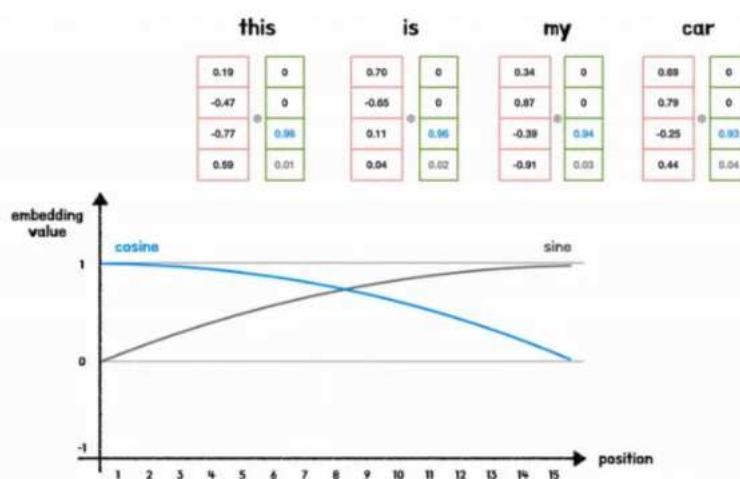
- 하지만 sine & cosine는 -1~1 사이를 주기적으로 반복하여 위치 벡터값의 차가 작지 않게 된다.

3. 같은 위치의 토큰은 항상 같은 위치 벡터값을 가지고 있어야 한다. 하지만 서로 다른 위치의 토큰은 위치 벡터값이 서로 달라야 한다.



- 예를 들어 1번쨰 토큰(position 0)과 9번째 토큰(position 8)의 경우 벡터값이 같아지는 문제가 발생한다.

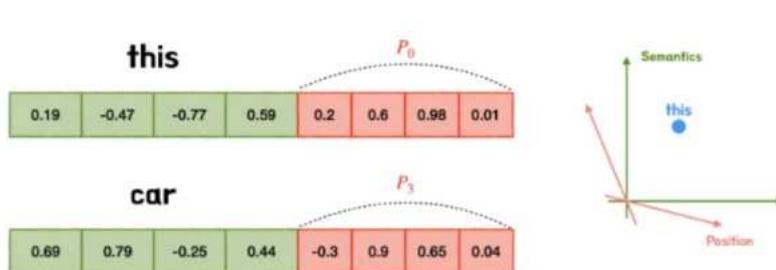
- 따라서 위치 벡터값이 같아지는 문제를 해결하기 위해 다양한 주기의 sine & cosine 함수를 동시에 사용한다.



- 위치 벡터가 4개의 차원으로 표현된다면, 각 요소는 서로 다른 4개의 주기를 갖게 되기 때문에 서로 겹치지 않는다.

3.5.4 Concatenate 대신에 Summation 연산을 사용한 이유





- `Concatenate`를 사용하면 단어의 의미 정보가 자체 차원 공간을 갖게 되고, 위치 정보 역시 자체 차원 공간을 갖으며, 직교성질(orthogonal)에 의해 둘은 서로 전혀 관계없는 공간에 있게 된다.
- 정보가 뒤섞이는 혼란을 피할 수 있게 해주지만, 메모리, 파리미터, 런타임 등과 관련된 비용 문제가 발생한다.
- `Summation`을 사용한다면 모델이 위치 정보를 적절하게 가지게 되는 동시에 단어 의미 정보 역시 충분히 강력하게 유지되어 벡터 공간에서 단어 의미 정보와 위치 정보 간의 거리가 적극해진다.

4. Why Self-Attention

recurrence, convolution과 비교했을 때 장점을 다음과 같이 정리할 수 있다.

- 하나의 layer 당 전체 연산 복잡도 감소
→ 별별 처리 연산의 양을 대폭 늘려 차인스레 학습 시간 감소
- Long term dependency의 문제점 해결
→ 토큰간의 물리적인 거리값들 중 최대값이 다른 모델이 비해 매우 불안정한 경우 같은 의존성을 잘 학습할 수 있다.
- 해석 가능한 모델
→ Attention이라는 가중치를 시각화하여 토큰들 간의 대용관계를 눈으로 직접 확인할 수 있다.
→ 다른 모델에 비해 Transformer는 모델이 내뱉은 결과를 해석할 수 있다.

.

SEARCH

단어 찾

- integral: 원수적인
- transduction: 전달
- preclude: - 못하게 하다, 불가능하게 하다(방해하다)
- To the best of our knowledge: 우리가 아는 한

REFER

	Time	Topic	Size
[1]			

