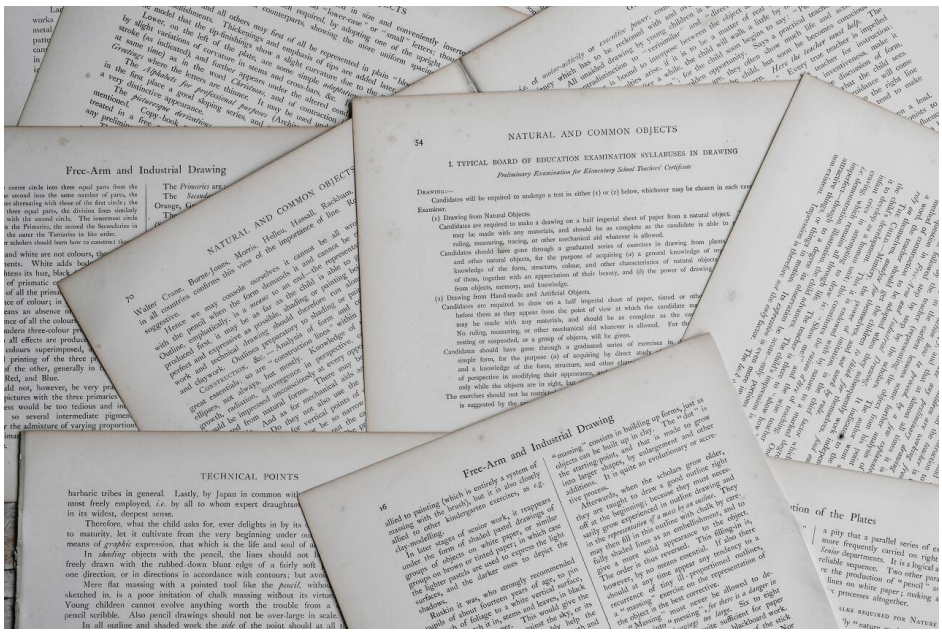
대학원 이야기/논문 리뷰

[논문 리뷰] (다시 읽어본) Attention Is All You Need (NIPS 2017)

by misconstructed 2020. 12. 3.

바닐라치과의원 치15018
임플란트,치아교정,치아미백,충치치료,감염관리기준 준수
http://drvanilla.co.kr



분류 전체보기

기술 면접

입시

관련 공부

논문 리뷰

CS224N : NLP with Deep Lear..

필기 노트

Python 기초

처음 배우는 딥러닝 수학

R을 이용한 통계학개론

자연어 처리 입문

Huggingface Transformers

잡동사니

NLP에 있어서 정말 큰 변화를 가져온 Transformer에 대한 논문이다. 과거에 논문을 읽어보고 나름대로 정리해본 글(여기)이 있는데, 아직 안읽어봤다면 읽어보지 않는 것을 추천한다. 최근에 어느 Github에서 제공한 Transformer의 전체적인 동작 구조와 코드를 이해하기 쉽게 정리한 것을 보고 내 나름대로 정말 간단하게 정리한 글도 있긴한데, 그닥 도음이 될 것 같지는 않다. (글, Github 은 한 번 둘러보는 것을 추천한다.)
논문의 원본은 여기에서 확인할 수 있다.

LSTM과 GRU 와 같은 **recurrent neural network**는 sequence modeling 분야 (대표적으로 Language modeling, machine translation emd) 에서 대부분의 state-of-the-art 성능을 제공하고 있다. Recurrent model은 input sequence 를 순차적으로 입력받는데, 입력받는 timestep t 에서 해당 입력과 이전 hidden state h_{t-1} 을 기반으로 현재 hidden state h_t 를 생성한다. 이렇게 모든 처리를 순차적으로 해야하는 특징 때문에, 여러 작업을 동시에 수행하지 못했고, 당연히 병렬처리의 효율도 떨어지게 되었다. 이러한 문제들을 해결하기 위해 최근에는 factorization trick이나 conditional computation 등의 방법들이 제안되었지만, 순차적으로 데이터를 처리한다는 큰 틀은 벗어나지 못하고 있다. 순차적으로 데이터를 처리할 때 발생하는 이러한 문제들을 해결하기 위해서 ByteNet, ConvS2S 등과 같이 convolutional neural network 를 사용하는 경우들도 있다. 하지만 CNN을 사용하는 경우, 입력의 길이가 길어질수록 입력의 원소와 출력의 원소 사이의 의존성을 학습하기 위한 distance가 매우 길어진다는 문제가 있다.
Attention mechanism은 sequence modling 을 할 때 입력과 출력의 원소들 사이의 거리와 무관하게 의존성(dependency)를 학습할 수 있다. 대부분의 경우 attention은 기존의 recurrent network와 함께 사용되어 왔다.
이번 논문에서는 **Transformer**를 제안한다. Recurrence를 제거함으로서 위에서 언급한 문제들을 해결하고, 오직 attention mechanism 만을 이용해서 입력과 출력 사이의 global dependency 를 학습하게 된다. Recurrence 가 없어졌으니 당연히 더 손쉽게 병렬처리가 가능해진다. 또한, CNN을 사용했을 때 문제였던 원소들 사이의 거리가 매우 멀어지는 문제도 상수개의 연산으로 고정시킬 수 있는 매우 큰 장점이 있다. Transformer의 가장 큰 특징 중 하나는 **self-attention**을 수행한다는 것이다. Self-attention은 동일한 sequence(입력, 출력 등) 내의 원소들 사이에서 attention을 수행하는 것이다. Self-attention은 Transformer에서 처음 사용된 것은 아니고, reading comprehension, abstractive summarization 등의 분야에서 많이 사용되고 있는 기법 중 하나이다.

공지 :) 구독하기

최근글 인기글

[Huggingfac Datasets… 2021.07.18

[Huggingfac PreTrained·· 2021.07.18

[Huggingfac Pretrained·· 2021.07.16

[Huggingfac PreTrained·· 2021.07.16

[Huggingfac Huggingfac· 2021.07.09

최근댓글

오늘부터 당신은 jour_be...

정말 흥미로운 주제네요😉

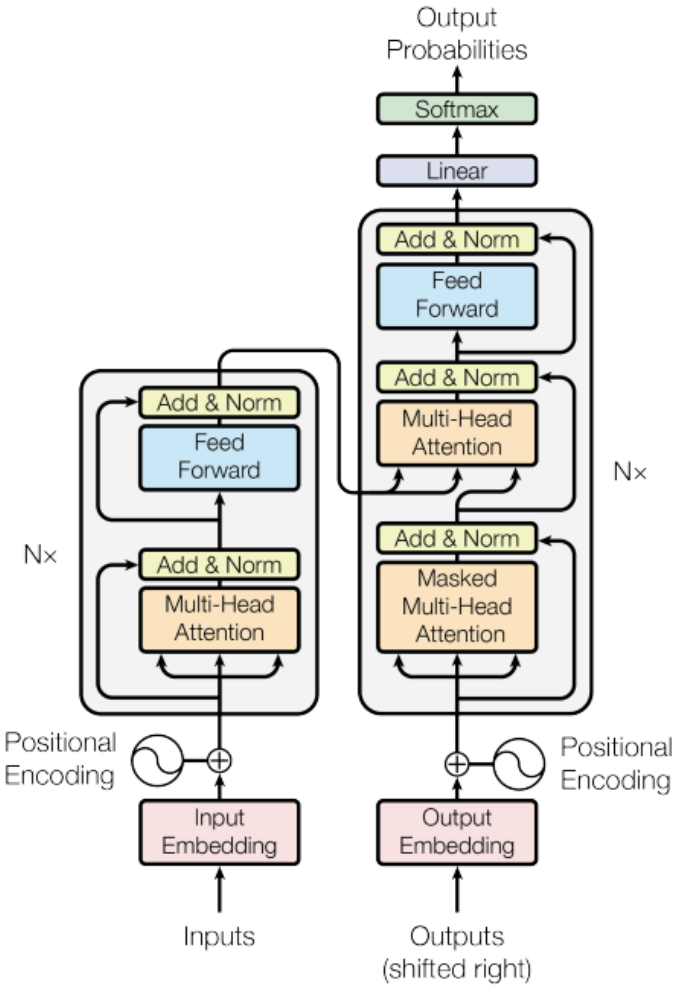
직업이 똑똑이 이신가요?!...

안녕하세요~ 한문제만 푸...

태그

transformer, 자료형, NLP, 파이썬 기초, seq2seq, CS224N, RNN, 딥러닝 수학, 인공지능, huggingface, 파이썬, MT, Python, 자연어 처리, 통계학 기초, Machine Translation, sequence to sequence, dialogue generation, 합격, natural language processing,

모델 구조



[그림-1]

Transformer는 stacked self-attention과 point-wise fully connected layer로 구성된 단순한 구조를 사용한다. Transformer의 **encoder**는 input sequence $(x_1, \ldots, x_n)$ 을 $z = (z_1, \ldots, z_n)$ 으로 매핑한다. **Decoder**에서는 한 번에 한 토큰씩 생성하는데, 이전 timestep에서 생성한 결과를 입력으로 사용해서 현재 timestep의 결과 토큰을 생성한다. 결과적으로 디코더에서도 output sequence $y = (y_1, \ldots, y_m)$ 을 생성한다. 인코더와 디코더의 역할은 다음과 같다.

- **Encoder** : 인코더는 총 N = 6 개의 층으로 구성되어 있다. 각 층은 2개의 sublayer로 구성되어 있다. 첫 번째 sublayer는 multihead self-attention layer이다. 두 번째 sublayer는 position-wise fully connected feed-forward network 이다. 각 sublayer의 입력을 결과에 더해주는 residual connection을 수행하고, 그 결과에 대해서 layer normalization을 수행한다. 모든 sublayer의 출력은 $d_{model} = 512$ 차원으로 구성된다. (디코더도 마찬가지)
- **Decoder** : 디코더도 동일하게 총 N = 6 개의 층으로 구성된다. 인코더와 거의 동일한 구조지만 마지막에 세 번째 sublayer가 추가되었다. 세번째

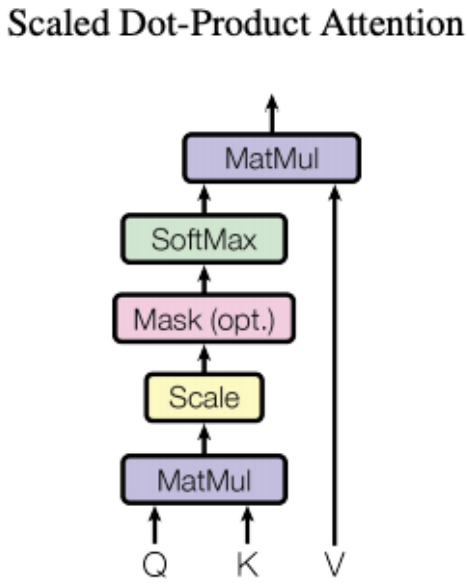
딥러닝, 후기, 인공지능, 기초, 통계학, 딥러닝 기초, Attention, Transformers, word2vec

:) 구독하기

전체 방문자
32,942
Today : 178
Yesterday : 81

sublayer는 인코더의 출력 결과 z 에 대해서 multi-head attention을 수행하는 것이다. 인코더와 동일하게 모든 sublayer의 결과에 residual connection과 layer normalization을 수행한다. 인코더와 또 다른 차이점 하나는 self-attention을 수행할 때, 미래의 내용에 대해 attention을 수행하지 못하도록 masking을 한다.

:) 구독하기



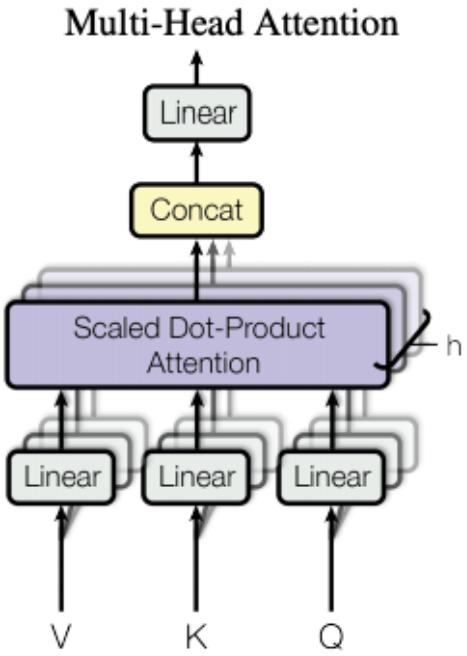
[그림-2]

Attention ([그림-2]) : Attention은 query 값을 key-value 쌍에 대해서 매핑하는 과정이라고 할 수 있다. 출력은 각 value들의 weighted sum으로 구성되는데, 이 때의 weight는 query와 key를 입력으로하는 함수에 의해서 구해진다. Transformer에서는 **"scaled dot-product attention**"을 수행한다. Query와 Key는 d_k 차원의 벡터로 제공되고, value는 d_v 차원의 벡터로 제공된다. Query와 Key의 연산은 우선, 1) Query와 Key의 dot-product를 수행, 2) $\sqrt{d_k}$ 로 dot-product 결과를 나눠주고, 3) softmax 함수를 통해서 확률값인 weight를 출력으로 제공하게 된다. ([식-1]) 가장 많이 사용되는 attention 방법으로는 additive-attention과 dot-product attention이 있는데, dot-product 연산이 연산 속도와 space-efficient 하기 때문에 dot-product attention을 사용하게 되었다. 하지만, 단순히 dot-product만 사용하는 경우 연산 결과가 매우 커질 수 있기 때문에, $\sqrt{d_k}$ 로 나눠주게 된다.

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

[식-1]

:) 구독하기



[그림-3]

Multi-head Attention ([그림-3]) : 단순히 query, key, value에 대한 single attention을 수행하는 것보다, d_{model} 차원의 벡터를 h **개**로 나눠서 각각 d_k, d_k, d_v 크기의 벡터들을 **병렬로 처리**하는 방법을 사용했다. 각 연산은 결과적으로 d_v 차원의 벡터를 제공하는데 모든 값을 다 구하면 다시 합쳐서 입력으로 제공한 d_{model} 차원의 벡터로 만들어 fully connected layer의 입력으로 제공한다. 이러한 방식은 각 representation의 서로 다른 위치에서 서로 다른 특징들을 학습하는데 도움이 된다. Muti-head attention의 연산을 [식-2]와 같다. 실제 모델에서는 $h-8$ 로 정해서 총 **8개의 head**를 생성하고, 각 head는 $d_k = d_v = d_{model}/h = 512/8 = 64$ 차원의 벡터로 구성한다. 연산 비용은 single attention과 거의 동일하다.

$$\begin{aligned}\text{MultiHead}(Q,K,V) &= \text{Concat}(\text{head}_1,...,\text{head}_\text{h})W^O \\ \text{where head}_\text{i} &= \text{Attention}(QW_i^Q, KW_i^K, VW_i^V)\end{aligned}$$

[식-2]

결과적으로 Transformer 에서는 총 3개의 서로 다른 multi-head attention을 수행한다.

1. **encoder-decoder** attention : 인코더의 결과가 key, value 역할을 하고, 디코더의 결과가 query 역할을 수행한다. 현재 디코더의 결과가 인코더의 모든 출력에 대해서 attention을 수행(의존성을 학습)할 수 있도록 한다.
2. **encoder** self-attention
3. **decoder** self-attention : 인코더의 self-attention과 동일하지만, 한 가지 다른점은 미래에 오는 값들에 대한 attention을 수행하지 못한다는 것이다.

현재 토큰 이후의 값들에 대해서는 masking out을 해서 접근하지 못하게 한다.

:) 구독하기

Self-Attention : Self-attention의 장점을 3가지 제시한다.

1. 각 층에서 발생하는 **computational complexity** 감소 : self-attention의 연산 속도는 입력의 길이 n 이 d_{model} 의 크기보다 작은 경우 더 빠르다. (대부분의 데이터셋을 구성하는 문장들은 d_{model} 보다 길이가 짧다.) 만약 더 긴 입력이 제공된다면, restricted self-attention을 수행할 수 있는데, 현재 단어에서 r 만큼 떨어진 단어들에 대해서만 attention을 수행하는 방식이다.
2. 동시에 **병렬**적으로 연산 가능
3. 멀리 떨어진 원소들 사이의 **path length** 감소 : sequence 를 다루는 문제에서는 멀리 떨어진 원소들 사이의 long-term dependency를 학습하는 것이 큰 도전 중 하나다. [표-1]을 보면, self-attention을 수행했을 때, 두 원소들 사이의 path length가 상수배로 매우 작은 것을 알 수 있다.

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)	$O(r \cdot n \cdot d)$	$O(1)$	$O(n/r)$

[표-1]

Position-wise Feed-Forward Network : 모든 인코더와 디코더 층에 Position-wise Feed-Forward Network가 존재한다. Position-wise Feed-Forward Network에서는 2번의 linear transformation을 수행하고, 두 linear transformation 사이에는 ReLU 연산을 수행한다. ([식-3]) 입력 벡터의 크기는 $d_{model} = 512$ 이고, 중간 출력 결과(W_1 과 연산한 결과 벡터) 는 $d_{ff} = 2048$ 차원으로 지정한다.

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

[식-3]

Positional Encoding : Transformer에서는 recurrence 와 convolution 연산을 수행하지 않기 때문에, 입력의 순서 정보를 유지하기 위해 positional encoding을 사용한다. 각 입력 토큰의 위치 정보를 sine 함수와 cosine 함수를 이용해서 [식-4] 와 같이 정하게 된다. 그냥 positional embedding을 함께 학습하는 방법도 실험해 봤는데, 거의 동일한 성능을 제공했다. [식-4]와 같은 방식을 선택한 이유는 학습 과정에서 접하지 못한, 더 긴 입력에 유연하게 대응할 수 있어서이다.

:) 구독하기

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

[식-4]

학습 세부사항

1. 데이터셋 : WMT 2014 English-German dataset (4.5million sent. pairs, byte-pair encoding), WMT 2014 English-French dataset (36M sent., 32000 word-piece vocab.)
2. 하드웨어 정보 : 8개의 NVIDIA P100 GPU를 사용해서 base model은 100,000 step (1step = 0.4초), big model 은 300,000step(1step = 1.0초) 동안 학습을 진행했다.
3. Optimizer : **Adam** optimizer 사용해서 learning rate는 [식-5] 와 같이 warmup step = 4000으로 두고 학습을 진행했다.
4. Residual Dropout : 각 sublayer의 결과에 대해서 residual connection을 수행하기 전에 dropout 을 적용한다. $P_{drop} = 1$ 으로 지정한다.
5. Label Smoothing : label smoothing value = 0.1 로 지정해서 사용한다. Perplexity 에는 부정적인 영향을 주지만, accuracy와 BLEU 점수에는 긍정적인 영향을 준다.

실험 결과

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**	$2.3 \cdot 10^{19}$	

[표-2]

[표-2]는 WMT 2014 English-German translation task(EN-DE) 와 WMT 2014 English-Frensh translation task(EN-FR) 의 결과다. 다 잘나온다. (SOTA를 뛰어넘는다.)

모델 변형

:) 구독하기

	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

[표-3]

위에서 제안한 Transformer의 구조에서 몇 가지 요소들을 변경한 후 English-German translation task에 적용한 실험 결과이다.

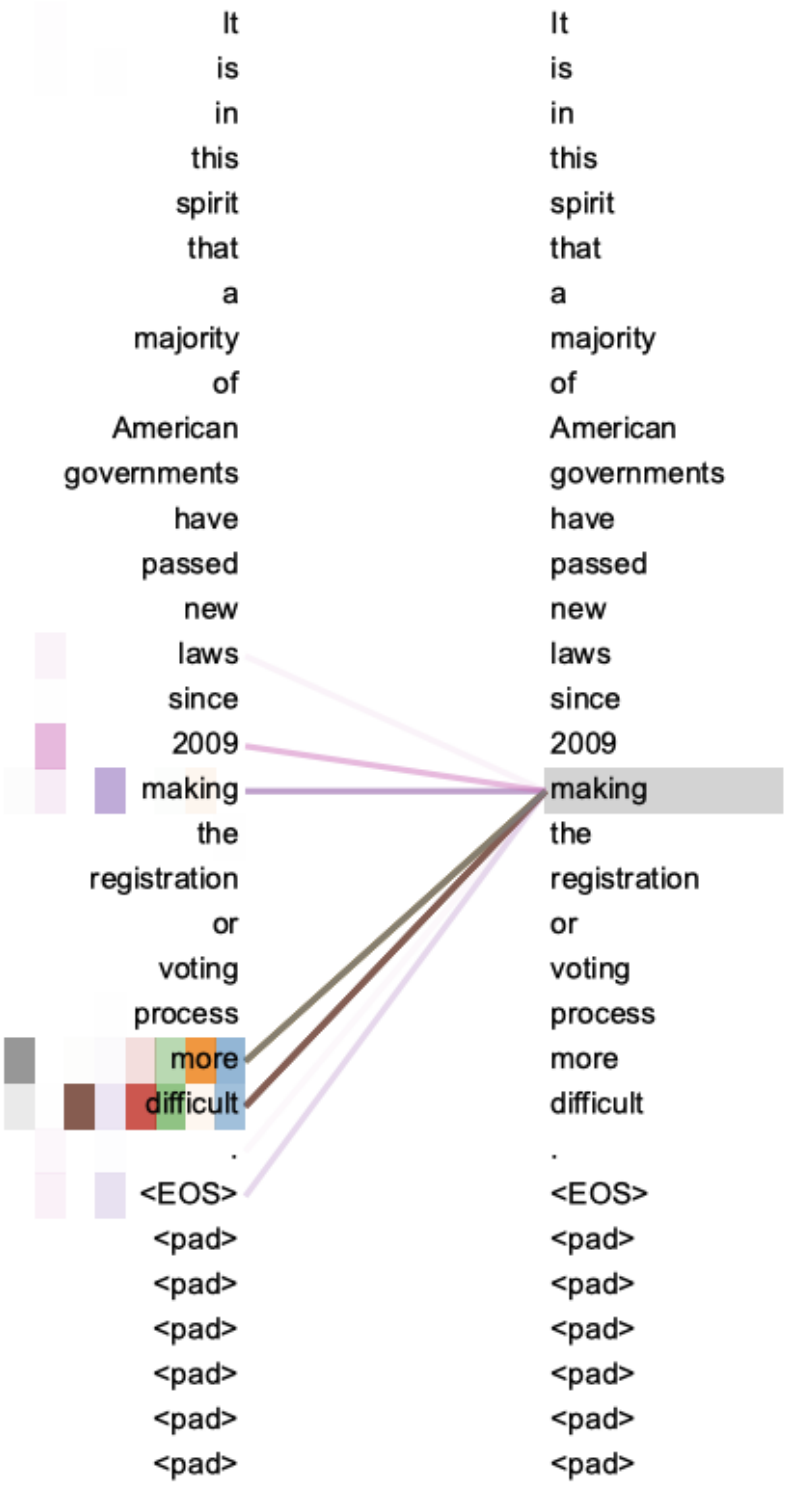
- (A) : head의 개수, d_k. d_v 값 변경한 경우 : head가 너무 많으면 오히려 성능이 떨어진다.
- (B) : d_k 만 변경
- (C) : 모델의 크기를 키운 경우 : 모델이 커지면 성능이 더 좋아진다.
- (D) : dropout 의 영향 : dropout 도 성능에 영향을 미친다.
- (E) : positional embedding의 중요성 : learned positional embedding을 사용해도 성능에 큰 변화는 없다.

다른 task에 대한 실험 : 다른 task에 대해서도 사용이 가능할지 실험하기 위해, English Constituency Parsing task에 transformer를 적용해봤다. 최고의 성능을 제공하지는 못하지만, 의외로 상당히 좋은 성능을 제공한다는 것을 확인할 수 있다. ([표-4])

Parser	Training	WSJ 23 F1
Vinyals & Kaiser el al. (2014) [37]	WSJ only, discriminative	88.3
Petrov et al. (2006) [29]	WSJ only, discriminative	90.4
Zhu et al. (2013) [40]	WSJ only, discriminative	90.4
Dyer et al. (2016) [8]	WSJ only, discriminative	91.7
Transformer (4 layers)	WSJ only, discriminative	91.3
Zhu et al. (2013) [40]	semi-supervised	91.3
Huang & Harper (2009) [14]	semi-supervised	91.3
McClosky et al. (2006) [26]	semi-supervised	92.1
Vinyals & Kaiser el al. (2014) [37]	semi-supervised	92.1
Transformer (4 layers)	semi-supervised	92.7
Luong et al. (2015) [23]	multi-task	93.0
Dyer et al. (2016) [8]	generative	93.3

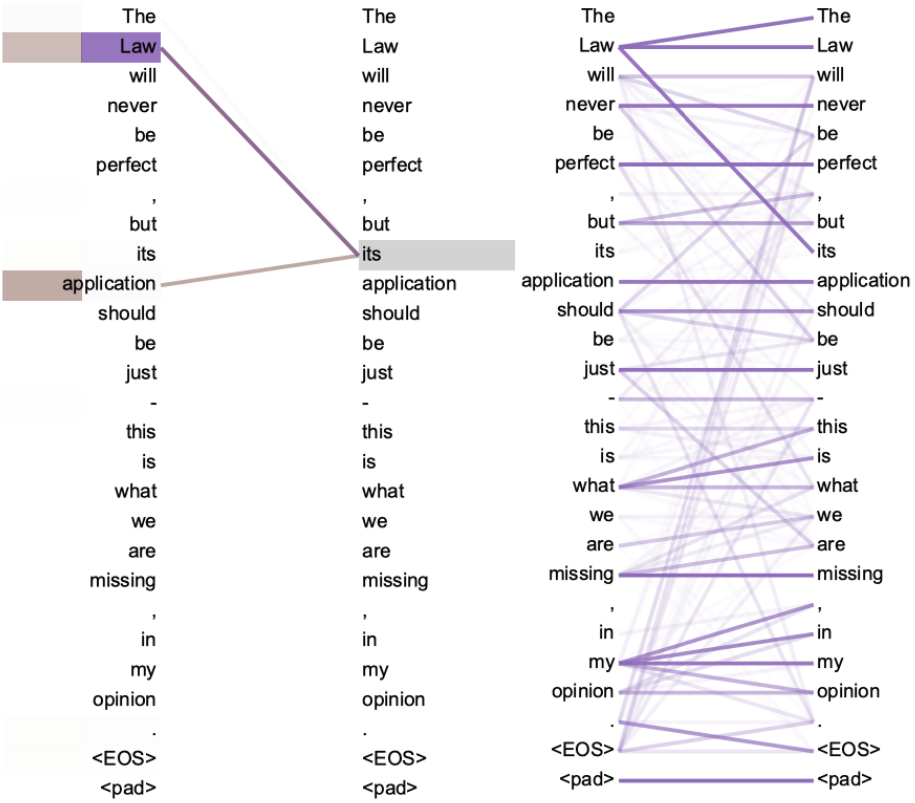
추가 자료 : Attention visualization

:) 구독하기

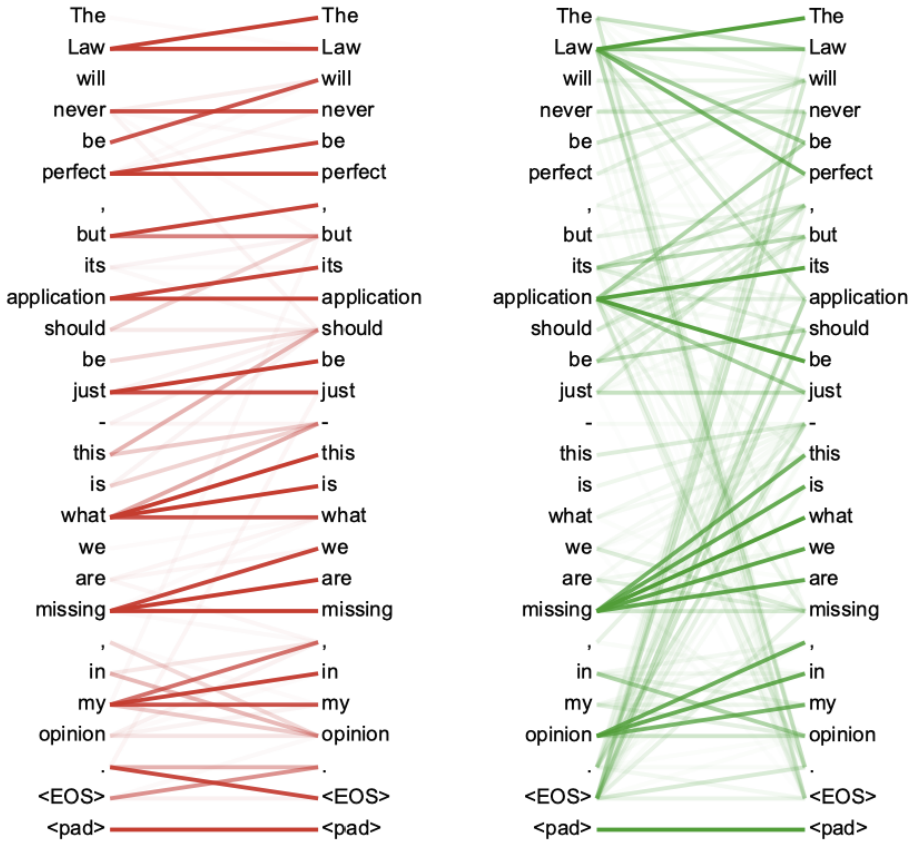

It
is
in
this
spirit
that
a
majority
of
American
governments
have
passed
new
laws
since
2009
making
the
registration
or
voting
process
more
difficult
.
<EOS>
<pad>
<pad>
<pad>
<pad>
<pad>
<pad>
It
is
in
this
spirit
that
a
majority
of
American
governments
have
passed
new
laws
since
2009
making
the
registration
or
voting
process
more
difficult
.
<EOS>
<pad>
<pad>
<pad>
<pad>
<pad>
<pad>

[그림-4]

:) 구독하기



[그림-5]



[그림-6]

참고 논문

1. Long short-term memory. : LSTM
2. Empirical evaluation of gated recurrent neural networks on sequence modeling.
3. Sequence to sequence learning with neural networks. : seq2seq (리뷰)
4. Neural machine translation by jointly learning to align and translate. : attention (리뷰)
5. Learning phrase representations using rnn encoder-decoder for statistical machine translation.
6. Google's neural machine translation system: Bridging the gap between human and machine translation. : word-piece
7. Neural machine translation of rare words with subword units. : byte-pair representation
8. Effective approaches to attention based neural machine translation.
9. Exploring the limits of language modeling.
10. Factorization tricks for LSTM networks. : factorization tricks
11. The sparsely-gated mixture-of-experts layer. : conditional computation
12. Structured attention networks.
13. Neural machine translation in linear time. : ByteNet
14. Convolutional sequence to sequence learning. : ConvS2S
15. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies : long-term dependency problem
16. Deep residual learning for image recognition. : residual connection
17. Layer normalization. : layer normalization
18. Adam: A method for stochastic optimization. : Adam optimizer
19. Dropout: a simple way to prevent neural networks from overfitting. : dropout

:) 구독하기

특가 2만원대 따뜻한 양털 롱코트!
STL

1 구독하기

'**대학원 이야기** > **논문 리뷰**' 카테고리의 다른 글

[논문 리뷰] How NOT To Evaluate Your Dialogue System: An Empirical Study of Unsupervised Evaluation Metrics for Dialogue Response Generation (ENMNLP 2016) (0) 2020.12.15

[논문 리뷰] Towards Empathetic Open-domain Conversation Models: a New Benchmark and Dataset (ACL 2019) (0) 2020.12.14

[논문 리뷰] (다시 읽어본) Attention Is All You Need (NIPS 2017) (0) 2020.12.03

[논문 리뷰] Listening between the Lines: Learning Personal Attributes from Conversations (WWW 2019) (1) 2020.11.25

[논문 리뷰] You Impress Me: Dialogue Generation via Mutual Persona Perception (수정 중) (ACL 2020) (0) 2020.11.23

[논문 리뷰] A Persona-Based Neural Conversation Model (ACL 2016) (0) 2020.11.19

:) 구독하기

태그

Attention | Attention is all you need | natural language processing | NLP | transformer

관련글


[논문 리뷰] How...


[논문 리뷰] Tow...


[논문 리뷰] Liste...


[논문 리뷰] You ...

댓글 0

이름	비밀번호
여러분의 소중한 댓글을 입력해주세요.	

비밀글 등록

TEL. 02.1234.5678 / 경기 성남시 분당구 판교역로
© Kakao Corp.