Attention is all You need

1. Introduction

순환신경망은 sequence modeling과 transduction problems의 접근 방식으로 사용되고 있으며 순환 언어 모델과 인코더-디코터 구조의 경계를 넓히려는 노력이 계속되고 있다.

- 순환 모델
  - 이전 hidden 층인 h_t-1와 위치 t에 대한 입력의 함수를 통해 숨겨진 층인 h_t의 시퀀스를 만든다. 이 경우 sequence의 길이가 길어지면서 메모리의 한계로 병렬화에 제약이 생긴다.
  - 최근 연구에서 factorization tricks와 conditional computation을 통한 computational efficiency을 향상하였지만 근본적인 제약이 남아있다.
- Attention mechanisms
  - 입출력 sequences의 거리와 관계 없이 modeling of dependencies를 가능하게 한다. 하지만 몇몇 경우를 제외하면 이 메커니즘은 순환망과 결합하여 사용된다.
- Transformer를 제안
  - 완전히 attention mechanism에 의존하는 모델 구조로 입력과 출력 사이의 dependencies 도출하고 훨씬 많은 병렬화를 가능하게 한다.

2. Background

- sequential computation를 줄이려는 목표는 the Extended Neural GPU, ByteNet ,ConvS2S의 기반이 되었다.
  - 합성곱 신경망을 기본으로 모든 입력과 출력의 위치에 대한 hidden representations을 병렬로 계산한다. 입력과 출력 위치가 멀면 연산의 수가 증가하여 dependencies를 배우기 어려워진다.
  - transformer의 경우 연산 수를 상수로 낮추지만 attention이 가중된 위치의 평균화로 인해 효과적인 resolution은 희생한다.
- Self-attention(called intra-attention)
  - sequence representation을 계산하기 위해 하나의 sequence의 위치들을 연관시키는데, 독해, 추상적 요약과 같은 다양한 분야에서 효과적으로 사용된다.
- End-to-end memory Networks
  - sequence-aligned 순환 대신 attention mechanism 순환을 기반으로 하였고 간단한 질의응답과 언어 모델링에 좋은 성능을 보인다.
- Transformer
  - 입력과 출력의 representation 계산에 self-attention에만 의존하는 첫번째 transduction model이다.

3. Model Architecture

인코더는 입력 시퀀스$(x_1, ..., x_n)$를 $z = (z_1, ..., z_n)$ 시퀀스로 매핑하고 $\mathbf{z}$가 주어지면 디코더는 한번에 하나의 element로 구성된 출력 시퀀스$(y_1, ..., y_n)$를 생성한다.이전에 생성된 symbol을 또다른 input으로 사용하여 그 다음 단계를 생성하면서 자기회귀한다.

Transformer는 stacked self-attention과 point-wise로 인코더와 디코더 모두 완전히 연결된 layers의 구조이다.

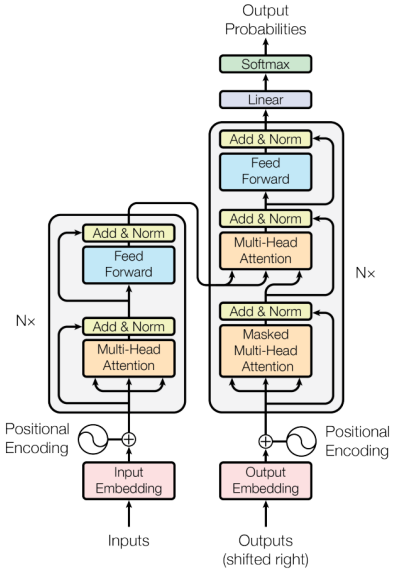


Figure 1: The Transformer - model architecture.

- Encoder and Decoder Stacks

  1. Encoder

  6개의 동일한 계층으로 구성되며 각 계층은 2개의 sub-layers를 가진다. 하나는 multi-head self-attention mechanism이며, 나머지는 position-wise connected feed-forward network이다.

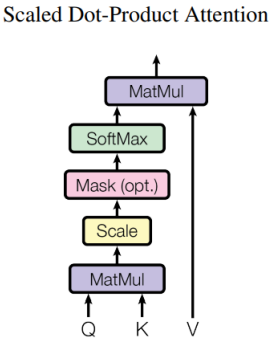
  각각의 sub-layer에 대한 잔차 연결한 후 계층 정규화를 하는데 이를 위해선 임베딩 레이어 뿐만 아니라 모든 sub-layers는 차원 d_{model} = 512라는 결과가 나와야 된다.

  2. Decoder

     6개의 동일한 계층으로 구성되고 2개의 sublayers에 더하여 인코더 스택의 결과에 대한 multi-head attention을 수행하는 세번째 sub-layer가 존재한다. 각 sub-layers에 대한 잔차연결 후 계층 정규화하고 디코더 스택의 self-attention sub-layer를 수정하여 위치가 다음 위치들까지 처리하는 것을 방지한다.

- Attention
  - query와 key-value 쌍의 집합을 output에 매핑하는데, 이때, query, keys, values, output은 모두 벡터이다. output은 value들의 가중된 합이며, 각 value의 가중치는 query와 query와 일치하는 key의 compatibility function으로 계산된다.

  1. Scaled Dot-Product Attention



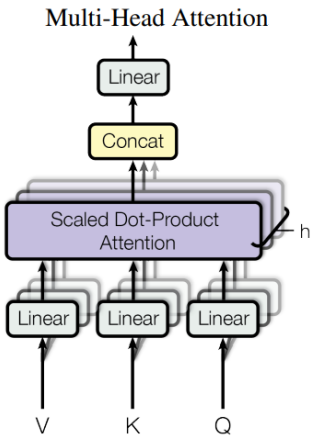
  - input : queries, 차원 d_k의 keys, 차원 d_v의 values

- 실제는 다음과 같이 계산되었다. queries의 집합으로 이루어진 matrix Q로 attention function 계산하고 keys는 matrix K, values는 matrix V를 구성한다.

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

- 가장 많이 쓰이는 attention functions은 2가지 이다. addictive attention은 하나의 hidden layer와 feed-forward network를 사용하여 compatibility function 계산하는 함수이다. dot-product attention는 최적화된 matrix 곱셈으로 수행될 수 있기 때문에 훨씬 빠르고 공간 효율적이다. scaling factor $\frac{1}{\sqrt{d_k}}$을 제외하고 본연구의 알고리즘과 동일하다.
- d_k의 커지면 dot-product 성능이 떨어지기 때문에 d_k에 softmax함수를 취해야해서 dot-products를 $\frac{1}{\sqrt{d_k}}$로 scale했다.

2. Multi-Head Attention



- queries, keys, values를 다른, 학습된 d_k, d_k and d_v 차원으로의 선형 투영으로 h번 각각 선형적으로 투영하는 것이 한번 학습하는 것보다 유익하다.

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

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

where the projections are parameter matrices $W_i^Q \in R^{d_{model} * d_k}, W_i^K \in R^{d_{model} * d_k}, W_i^V \in R^{d_{model} * d_v} and W^O \in R^{hd_v * d_{model}}$

h=8인 평행 attention layers(heads)를 사용하였는데, 각각의 layer에서 $d_k = d_v = d_{model}/h = 64$이다. 계산 비용은 모든 차원을 가진 single-head attention과 비슷하다.

3. Application of Attention in our Model

   3가지 방법으로 multi-head attention 수행하였다.

   1. encoder-decoder attention 계층에서 queries는 이전 디코더 계층에서, memory keys와 values는 인코더의 output에서 나오기 때문에 디코더의 모든 위치는 input 시퀀스의 모든 위치에 attend할 수 있다. 이는 seq2seq 모델에서 전형적인 encoder-decoder attention mechanisms을 모방한 것이다.
   2. 인코더는 self-attention 계층을 포함하는데, 이 계층에서 keys, values, queries는 같은 공간인 인코더의 이전 계층의 output에서 나온다.
   3. 자기 회귀를 위해 디코더에서 왼쪽으로 가는 정보의 흐름을 막아야 한다. 따라서 scaled dot-product attention에서 softmax의 입력에서의 모든 불법적인 연결인 values를 masking하여 수행한다.

- Position-wise Feed-Forward Networks
  - 인코더와 디코더에서 계층은 완전히 연결된 feed-forward network가 있으며 각 포지션에 따로, 동일하게 적용된다. 이 네트워크는 두 선형 transformation과 그 사이의 ReLU 활성화로 이루어진다.

    $FFN(x) = max(0, xW_i + b_1)W_2 + b_2$
  - 선형 transformation은 모든 위치에서 같지만, 계층마다 다른 parameters를 사용한다. 즉, kernel 사이즈가 1인 2개의 convolutions으로 입력과 출력의 차원 d_{model} = 512이며, inner-layer는 d_{ff}=2048이다.
- Embeddings and Softmax
  - 입력 토큰과 출력 토큰을 d_{model} 차원의 벡터로 변환하는 임베딩을 사용한다. 일반적으로 학습된 linear transformation과 softmax function을 통해 디코더 출력을 예측된 next-token 확률로 변환한다 . 2개의 임베딩 계층과 pre-softmax linear transformation 사이에서 같은 가중치의 행렬을 공유한다. 임베딩 계층에서는 가중치를 $\sqrt{d}_{model}$로 곱한다.
- Positional Encoding
  - 모델이 sequence의 순서를 사용하기 위해선 sequence에 토큰의 상대적이거나 절대적인 위치 정보를 투입하고 마지막엔 인코더와 디코더 stacks의 바닥에 있는 출력 임베딩에 'positional encodings'를 더하는데, 임베딩과 동일한 차원d_{model}을 갖기 때문에 둘을 합할 수 있다. 모델에서 positional encodings로 다른 frequencies의 사인과 코사인 함수를 사용한다. (pos: 위치, i:차원)

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

    $PE_{pos,2_{i+1}} = cos(pos/10000^{2i/d_{model}})$
  - 모든 고정된 offset k와 PE_pos+k는 PE_pos의 선형함수로 표현될 수 있기 때문 모델이 상대적인 위치로 쉽게 attend를 배울 수 있다고 가정하여 해당 함수를 선택하였다. 모델이 train 중 접한 것보다 더 긴 sequence 길이를 추정하게 하는 사인 곡선을 선택하였다.

4. Why Self-Attention

- self-attention을 사용하게 하는 3가지
  1. 계층 당 모든 계산 복잡성
  2. 병렬화될 수 있는 연산의 양(=요구되는 sequential 연산의 최소 수)
  3. 네트워크에서 긴거리 종속성 사이의 경로 길이

     입력과 출력 sequence에서 위치의 결합 사이에 경로들이 짧을수록 긴거리 종속성을 배우기 쉬워진다.

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 계층
  - 연산들에서 하나의 상수와 모든 위치를 연결한다. 시퀀스 길이 n이 차원의 수 d보다 작을 때 계산 복잡성 면에서 recurrent보다 빠르다. 기계 번역의 최신 모델에서 흔히 사용된다.
- convolution 계층
  - 커널 너비 k가 n보다 작으면 입력과 출력의 위치 쌍이 모두 연결되지 않고 두 위치 사이의 가장 긴 경로의 길이가 늘어나게 되며 recurrent보다 더 비용소모적이다.

5. Training

- Training Data and Batching
  1. data1 : 450만개의 문장쌍으로 이루어진 standard WMT 2014 English-German 데이터셋
     - 문장은 byte-pair 인코딩(37,000개의 토큰의 공유 소스 target 어휘)통해 인코딩 되어있음
  2. data2: 360만개의 문장과 32000 word-piece 어휘로 split된 토큰
- Hardware and Schedule
  1. GPU : 8 NVIDIA P100 GPUs
  2. basic models : 하나의 학습 단계 당 0.4초이며 총 10만개의 단계를 12시간 동안 학습
  3. big models : 단계마다 1초이며 총 30만 단계로 3.5일 동안 학습
- Optimizer
  1. Adam optimizer : $\beta1 = 0.9$, $\beta2 = 0.98$ and $\epsilon = 10^{-9}$
  2. 학습률

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

     warmup-steps에서 학습률은 선형적으로 증가하고 이후 단계 숫자의 역제곱근에 비례하여 감소한다.

- Regularization
  1. Residual Dropout

     각 서브계층의 출력이 서브계층의 입력에 더해지고 정규화되기 전에 인코더와 디코더 stacks 모두 임베딩과 위치 인코딩의 합에 dropout을 적용한다.

  2. Label Smoothing

     적용하면 모델이 확실하지 않은 방향으로 학습하여 혼란이 가중되지만 정확도와 BLEU score를 향상시킨다.

6. Results

- Machine Translation

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

  - base model은 10분 간격으로 쓰인 마지막 5개의 체크포인트를 평균하여 얻어진 모델을 사용하였고, big model은 마지막 20개의 체크포인트를 고려하였다.
  - WMT 2014 English-to-German translation task와 WMT 2014 English-to-French translation task에서 모두 transformer(big)은 ensemble을 포함한 이전 모델보다 더 나은 BLEU 성능을 보이며 학습 비용도 적다.

- Model Variations

	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

(A) head가 하나인 경우는 16개인 경우보다 BLEU가 낮았고 32개인 경우에도 더 낮았다. (b) key 크기 d_k가 낮아지면 모델의 품질도 떨어진다.

(C, D) 모델이 클수록 성능이 좋고 과적합을 예방하기 위해선 dropout이 도움이 된다. (E) 사인 곡선 위치 인코딩을 학습한 위치 임베딩으로 대체할 때, base model과 비슷한 성능을 보였다.

- English Constituency Parsing

''

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

  - 4-layer transformer 학습에서 차원 d_{model}=1024이고, train 데이터로는 Penn Treebank의 WSJ 부분, 약 4만개의 문장, 1만 6천개의 토큰을 사용하였다.
  - 준지도학습 1.7백만개의 문장을 가진 BerkleyParser corpora로 3만 2천개의 토큰을 사용하였다.
  - 실험 결과, 업무 특화된 튜닝 없이 wsj 훈련 데이터 만으로 transformer는 RNN seq-2-seq 모델[37]보다 더 나은 성능을 보여주었다.

7. Conclusion

Transformer는 완전히 attention만을 기반으로 한 첫번째 sequence transduction model로 인코더-디코더 구조에서 가장 흔히 사용되는 recurrence layer를 multi-headed self-attention으로 대체하였다.

translation task에서 transformer은 recurrent나 convolutional 계층 기반의 구조보다 훨씬 빨랐으며 앙상블 모델을 포함한 이전 모델들보다 더 나은 성능을 보여줌