Transformer

Abstract

우세한 시퀀스 번역은 복잡한 rrecurrent이거나 convolution 네트워크의 인코더와 디코더를 기반으로 하고 있다. 가장 좋은 모델은 어텐션 방식을 통한 인코더와 디코더를 연결하는 모델이다. 우리는 세로운 간단한 recurrence와 convolution방식을 전적으로 제가하며 self attention mechanism의 방식을 가지는 Transformer를 제안한다. 우리는 영어-독일 번역과 영어-프랑스 번역 테스크에서 가장 좋은 성능을 가졌으며 일반적으로 다른 테스크에서도 좋은 성능을 보임을 확인할 수 있었다.

Introduction

RNN과 LSTM 그리고 GRNN은 시퀀스 모델링과 번역 문제에서 좋은 성능을 내고 있다. 그리고 reccurent와 encoder-decoder방식에서 많은 노력들이 진행되어 왔다.

반복적인 모델(recurrent)은 입력 및 출력 순서에 따라 계산된다. 순서를 따라가며 계산하며 hidden states ht를 생성하기 위해 hidden states $ht-1$과 포지션값을 인풋으로 사용한다. 이러한 상속적인 시퀀셜은 학습과정에서 병렬을 수행하지 못하게 될 뿐만 아니라 긴 시퀀스에서 메모리 제약을 받게 된다. 이러한 효율성 문제는 여러 기법을 통해서 해소되었지만, 여전히 문제로 남아있다.

Attention은 다양한 테스크에서 인풋과 아웃풋의 길이를 무시하는 시퀀스 모델링과 번역 모델의 부분이 되었다. 몇몇 경우에서는 반복 모델의 일부로 결합되어 사용되었다.

우리는 attention기법을 통해 반복과 단기 의존성을 피하는 Transformer를 제시한다. Transformer는 병렬적 그리고 적은 학습으로 최고의 성능을 도달할 수 있다.

Background

연속적인 계산을 줄이기 위해 컨볼루션 네트워크를 사용하여 인풋과 아웃풋에 대해 병렬적으로 히든 representation을 계산했습니다. 이러한 모델은 인풋과 아웃풋에 위치에 따라 연산량을 증가시킵니다. 이는 거리에 따른 의존성을 더 어렵게 만들었습니다. Transformer에서는 연산을 줄였지만, averaging attention-weighted position으로 인해 효과적인 해상도가 감소했븝니다. 우리는 이를 Multi-head attention으로 대체했습니다.

intra-attention이라고도 불리는 Self-attention은 다른 위치에 있는 시퀀스끼리의 관계를 표현하고자 합니다. Self-attention은 글 이해, 요약등등 다양한 테스크에서 성공적이었습니다. Transformer는 RNN과 CNN이 없는 인풋과 아웃풋의 관계에 대해 계산하는 self-attention에 의존한 첫번째 모델입니다.

Model Architecture

가장 좋은 언어 모델은 인코더와 디코더의 구조를 가집니다. 인코더에 $x1, x2, x3$ ...의 인풋을 사용하면 인코더가 이를$z1, z2, z3$,... 로 표현하며 이는 디코더에 들어가 output인 $y1, y2, y3 \ldots$를 생성하게 됩니다. 모든 단계에서 모델은 자동적으로 회귀하며, 다음 것을 생성할 때 인풋 데이터를 집어넣습니다.

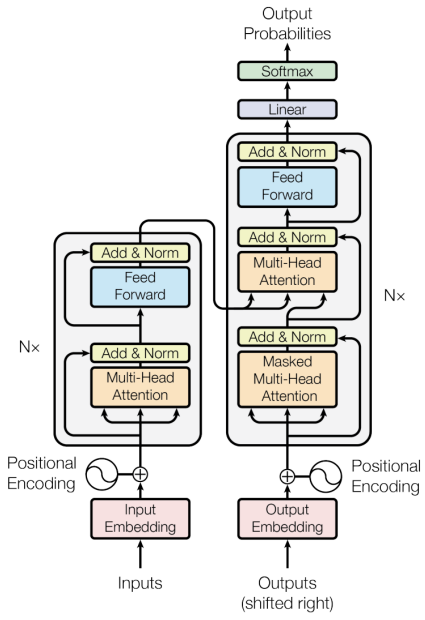


Figure 1: The Transformer - model architecture.

트랜스포머는 위와 같은 구조를 가지게 됩니다.

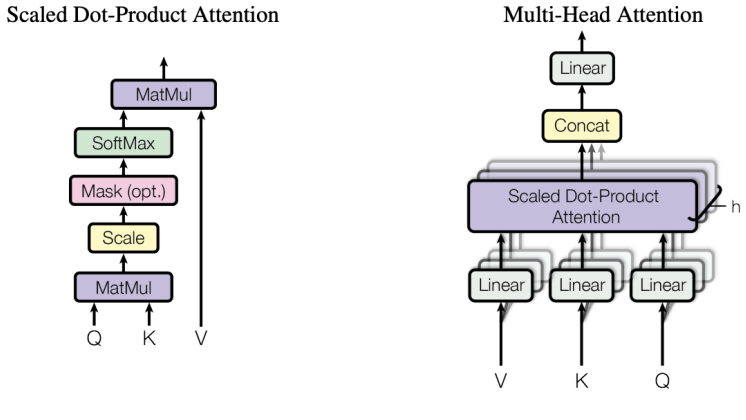
Encoder and Decoder Stacks

Encoder는 6개의 레이어 스택으로 이루어져있다. 각각의 레이어는 두개의 서브레이어를 가진다. 첫 번째는 multi-head self-attention mechanism이고, 두 번째는 position wise fully connected feed-forward networkdlek. 우리는 매 레이어의 residiual connection을 이용했고 그 뒤로 layer normalization을 사용한다. 그러면 각각의 레이어의 아웃풋은 LayerNorm(x+Sublayer(x))가 된다. 우리는 residual connection을 이용하기 위해서 모든 레이어와 임베딩의 아웃풋은 512로 설정하였다.

Decoder 또한 6개의 스택으로 이루어진다. 게다가 인코더의 두개의 레이어에 디코더는 인코더의 결과값에 대하여 multi-head attention을 수행하는 3번째 레이어를 삽입하였다. 인코더와 유사하게 residual connection과 layer normalization을 매 레이어 이후에 실행하였다. 우리는 또한 self-attention을 수정하여 이어지는 포지션에 의하여 포지션이 정해지는 것을 예방한다. 이 마스킹을 통해서 이전의 정보에 대해서만 의존할 수 있게 된다. (미래를 예측하기 위해서 미래의 정보를 배제하는 것.)

Attention

Attention은 벡터로 출력되는 query, key, value로 이루어 진다. 아웃풋은 query와 key로 부터 나오는 values의 가중합으로 이루어진다.



왼쪽이 Scaled Dot-Product Attention이다. 인풋의 구조는 dk의 차원을 가지는 query와 key이며, value는 dv의 차원을 갖게 된다. 우리는 query와 모든 keys에 대해서 내적을 진행하며, 각각 dk로 나누게 된다. 그리고 softmax function을 적용시켜 value 값을 구한다. Attention은 다음과 같은 식으로 표현이 가능하다.

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

대체로 additive attention과 dot-product attention이 사용되어진다. Dot-product attention은 우리의 알고리즘이며 scaling 요인인 $1/\sqrt{dk}$는 제외하고. additice attention은 Feed-forward network를 사용해서 호환성을 계산한다. 두가지의 attention은 비슷하고 이론적으로 복잡하나 dot-product attention은 행렬곱으로 표현이 되기 때문에 훨씬 빠르고 공간적으로 효율적이다.

Value 값이 작을 때 두개의 메카니즘은 유사하게 작동하지만 additive attention은 큰 값을 가지는 dk가 scaling이 되지 않을 때 더욱 효과적이다. 우리는 큰 값의 dk가 규모에서 dot product가 커지며 softmax function의 값이 극단적으로 가며 기울기 소실 문제를 일으킨다고 의심했고 우리는 이를 $1/\sqrt{dk}$로 scaling하여 대처하였다.

Single attention 대신 우리는 매번의 h time마다 다른 query, key, value값이 개별적으로 있는게 효과적이란것을 알아냈다. 이는 병렬적으로 수행하며 동일한 차원의 value값을 낸다. 이렇게 나온 결과값은 concat되어지며 다시 다른 레이어로 들어가게 된다.

여기서 우리는 h=8의 8개의 병렬적 레이어를 사용했다. 우리는 $dk = dv = d(model)/h = 64$를 사용한다. 이 때문에 각 헤드에서 차원이 감소되며 전체적인 연산량은 single-head attention과 유사해진다.

The Transformer uses multi-head attention in three diffrent ways

encoder-decoder attention layer에서 query는 이전의 decoder layer에서 오며 key와 value는 encoder로부터 온다. 이는 decoder에 모든 sequence에 대해서 수행되어진다. 이것은 전형적인 encoder-decoder attention을 모방한 것이다.

이러한 encoder는 self-attention layer를 포함하며 모든 key, value, query는 같은 장소에서 온다. 이 경우 이전 layer의 아웃풋을 말한다. 각각의 encoder에 위치는 이전 encoder의 모든 위치에 대해 학습하게 된다.

유사하게 decoder에 있는 self-attention layer는 decoder의 모든 위치에 대해 학습할려는 경향이 있고 우리는 이러한 정보 흐름을 방지해야했다. 그래서 우리는 마스킹처리를 한다.

Position-wise Feed-Forward Networks

모든 encoder와 decoder는 feed-forward network를 연결한다. 이는 두 개의 linear layer와 사이에 ReLU 활성화 함수가 포함 되어 진다. 인풋과 아웃풋은 512차원이며 중간에 있는 hidden layer는 2048차원을 가지게 된다.

Embedding and Softmax

다른 시퀀스 번역 모델과 유사하게 우리는 인풋 토큰과 아웃풋으로 변화하는 학습된 임베딩을 사용한다. 우리는 또한 학습된 linear와 softmax funtion을 사용한다. 우리 모델에서 우리는 두개의 임베딩 레이어와 softmax transformation간 같은 weight를 공유한다.

Positional Encoding

우리의 모델은 반복과 컨볼루션이 없기 때문에 모델이 sequence의 순서를 알 수 있도록 위치에 대한 정보를 주입해야한다.

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

우리는 이를 위해 positional encoding를 encoder와 decoder에 더해준다. positional encoding은 $d(moddel)$과 같은 차원을 갖게 되며 이 둘은 더해진다. 위 식에서 pos는 해당 token의 sequence이며 i는 차원을 뜻한다. 이러한 position encoding은 모든 위치를 표현할 수 있으며 위치에 대해 쉽게 학습할 수 있게 해준다. 삼각함수를 사용하여 길이가 긴 sequence데이터도 학습이 가능하다.

Why Self-Attention

1. layer 마다의 연산량 감소

convolution, recurrent에서 attention학습 방법으로 전환하며 layer내에서의 연산량을 감소 시킬 수 있다.

2. 병렬 수행 가능

연속적으로 진행되는 학습이 아닌 개별적인 연산이 따로 가능하여 병렬 수행이 가능하다.

3. 장기 의존성

각각의 토큰이 모든 토큰관의 관계에 대해서 학습을 하기 때문에 장기 의존성을 해결한다.

Result

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

Transformer는 성능적으로 가장 우수했으며 기존 model보다 연산량을 감소시켰다. 이전 기본 모델들을 압도했을 뿐만 아니라 앙상블을 진행한 모델들 보다 성능이 좋았다.

우리는 마지막으로 5개의 checkpoint를 평균을 내어 기본 model로 사용하며 big model로는 마지막 20개의 checkpoint를 평균 내어 사용한다. 또한 output length는 input length의 + 50개를 제한으로 두며 가능하다면 일찍 종료한다. 또한, beam을 4로 두어 진행한다.

	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

transformer의 parameter를 여러가지로 변환하며 실험을 하였는데 head가 무조건 적으로 많았을 때는 좋지 않았다. key의 차원, dropout, positional encoding등을 바꿔 가면 실험하였다.

Conclusion

우리는 처음으로 완전히 attention에 근거한 sequence model을 보여준다. reccurent layer를 대체하였고, encoder-decoder 구조를 multi-headed self-attention과 같이 사용하였다. Transformer는 이전 구조들보다 더 빠르게 학습이 가능했으며 더 높은 성능을 보여주었다.