boostcamp ai tech

Attention Is All You Need

Conference : NIPS 2017

citation : 70014

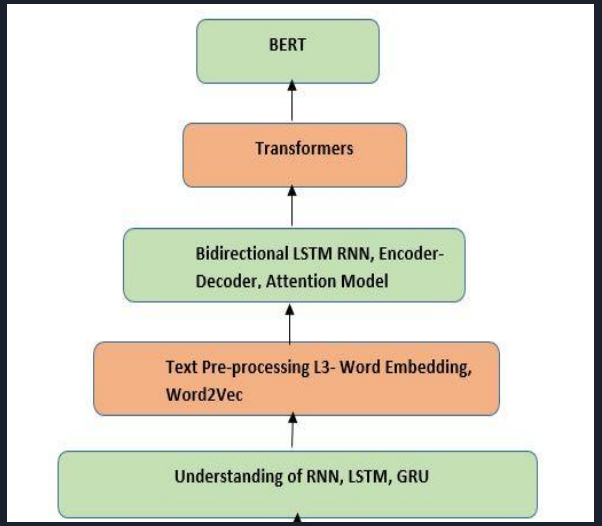
Authors :

- Ashish Vaswani (Google Brain)
- Noam Shazeer (Google Brain)
- Niki Parmar (Google Research)
- Jakob Uszkoreit (Google Research)
- Llion Jones (Google Research)
- Aidan N. Gomez (University of Toronto) • Łukasz Kaiser (Google Brain)
- Illia Polosukhin

목차

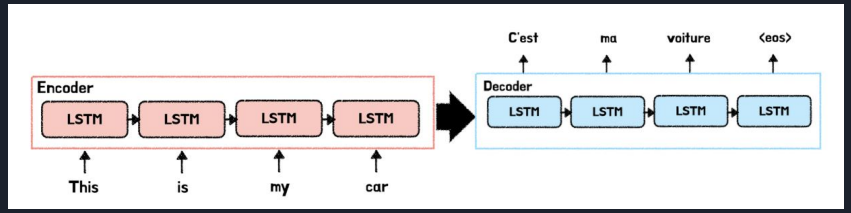
1. 도입

- 논문 이후 대부분의 아키텍처는 Transfomer 기반
  -> 패러다임의 전환
- Foundation Model



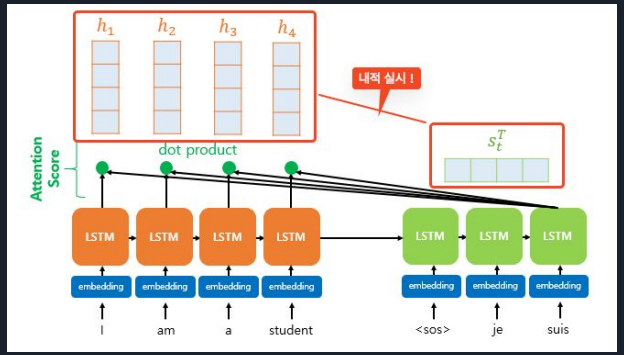
참고 자료 : Introduction to Natural Language Processing and Tokenization

1. 도입



Seq2Seq

- Encoder, Decoder
- Context Vector -> 정보의 손실
- 하나의 고정된 크기의 벡터에 모든 정보를 압축 시도 -> 정보 손실
- 병행 학습 불가 -> GPU 활용 능력이 떨어짐.



Attention

- 어텐션의 기본 아이디어는 디코더에서 출력 단어를 예측하는 매 시점(time step)마다, 인코더에서의 전체 입력 문장을 다시 한 번 참고 하자는 것.
- 구성요소
  - Query : 입력 시퀀스에서 관련된 부분을 찾으려고 하는 정보 벡터
  - Key : 관계의 연관도를 결정하기 위해 query 와 비교하는데 사용되는 벡터
  - Value : 특정 key에 해당하는 입력 시퀀스의 정보로 가중치를 구하는데 사용되는 벡터
- Attention Score
  - 현재 디코더의 시점 t에서 단어를 예측하기 위해, 인코더의 모든 은닉 상태 각각이 디코더의 현 시점의 은닉 상태와 얼마나 유사한지를 판단하는 스코어값

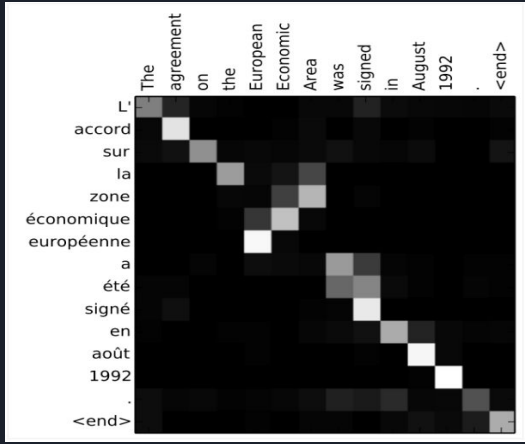
$$score(s_t,\ h_i) = s_t^T h_i$$

$$e^t = [s_t^T h_1, \ldots, s_t^T h_N]$$

[그림 자료 출처] 딥러닝을 이용한 자연어 처리 입문 15장 Seq2Seq 에서의 Attention Mechanism

1. 도입

- Attention
  - 어텐션 함수는 주어진 '쿼리(Query)'에 대해서 모든 '키(Key)'와의 유사도를 각각 구함.
  - 구해낸 이 유사도를 키와 맵핑되어있는 각각의 '값(Value)'에 반영
  - 유사도가 반영된 '값(Value)'을 모두 더해서 리턴



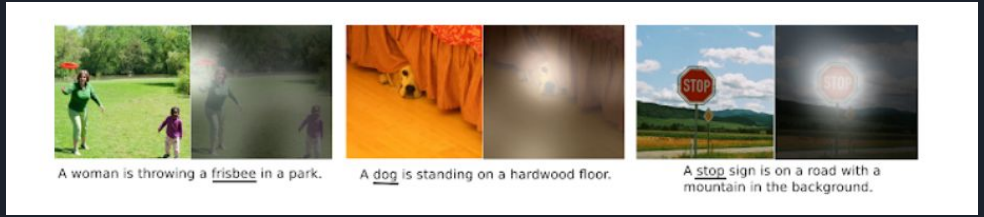


그림 출처
그림출처 2

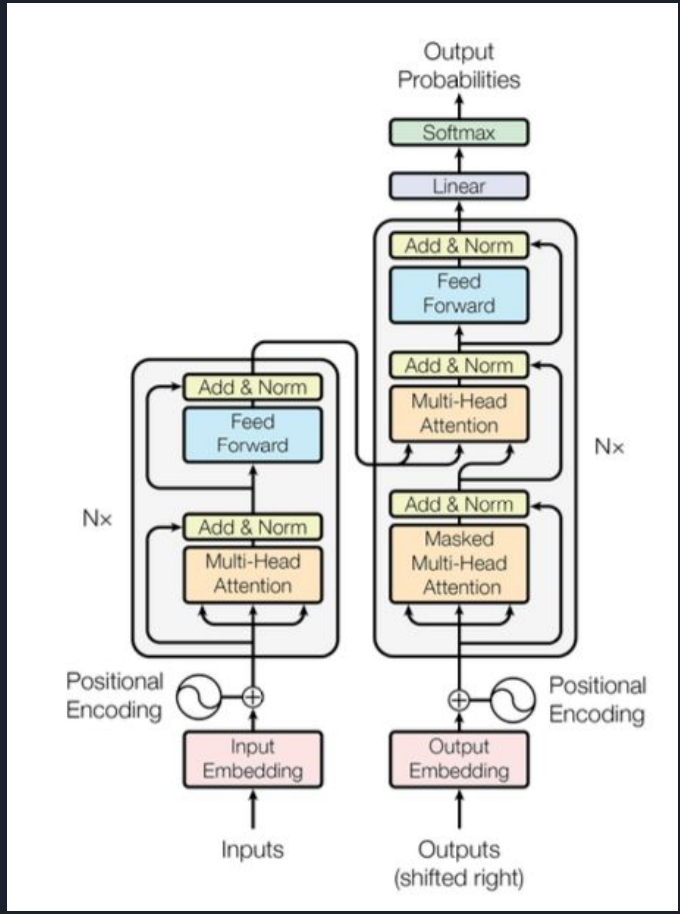
2. Transformer Architecture

Transformer

- Encoder - Decoder 모형
- 오직 Attention 만 사용한 모델

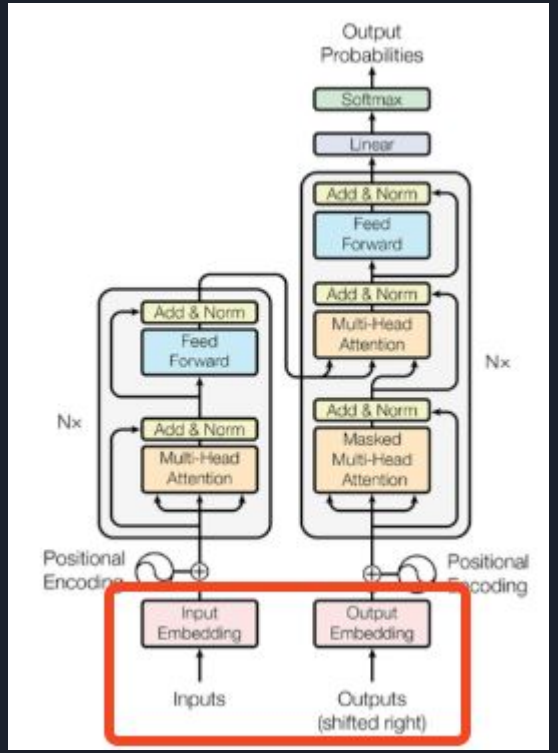
주요 구성

1. embedding
2. Positional Encoding
3. Self Attention
4. Multi-head Attention
5. Add & Norm
6. Masked Multi-Head Attention
7. Encoder - Decoder Attention



2.1 Embedding

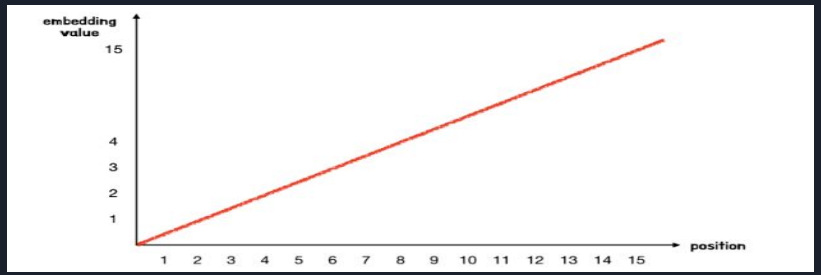
- “I like pizza”
- Corpus (1, 2, 3)
- Embedding, Look-Up Table
  - I : [0.1, 0.2, 0.3]
  - like : [0.4, 0.5, 0.6]
  - pizza: [0.7, 0.8, 0.9]



2.2 Positional Encoding

- RNN, LSTM 은 input에 입력되는 순서대로 처리
- 트랜스포머는 행렬곱으로 병렬로 한번에 처리한다. -> 입력 순서를 어떻게 처리 할 것인가?
- 두 문장은 순서를 고려하지 않았을 때 완벽하게 일치한다.
  - I am not a student, you are a teacher
  - I am a student, you are not a teacher
- 직관적 생각
  - 선형 함수를 사용하면 되지 않는가? -> 값 보정 필요
  - 문장의 길이를 잘라서 부여하면 되지 않을까? -> 길이에 따라 다름
- 주기성을 가지는 벡터를 기존 embedding 벡터에서 더해줘서 때문에 위치를 추적

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



2.2 Positional Encoding

- pos : positon index
- i : the dimension of the embedding
- d_model : dimension of the model

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

“ I like pizza”

I : [0.1, 0.2, 0.3, 0.4]

like : [0.5, 0.6, 0.7, 0.8]

pizza: [0.9, 1.0, 1.1, 1.2]

2.2 Positional Encoding

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

Position 1 (I):

PE(1, 0) = sin(1 / 10000^(0 / 4)) ≈ 0.841

PE(1, 1) = cos(1 / 10000^(1 / 4)) ≈ 0.999

PE(1, 2) = sin(1 / 10000^(2 / 4)) ≈ 0.002

PE(1, 3) = cos(1 / 10000^(3 / 4)) ≈ 1.000

Position 2 (like):

PE(2, 0) = sin(2 / 10000^(0 / 4)) ≈ 0.909

PE(2, 1) = cos(2 / 10000^(1 / 4)) ≈ 0.997

PE(2, 2) = sin(2 / 10000^(2 / 4)) ≈ 0.005

PE(2, 3) = cos(2 / 10000^(3 / 4)) ≈ 0.999

Position 3 (pizza):

PE(3, 0) = sin(3 / 10000^(0 / 4)) ≈ 0.141

PE(3, 1) = cos(3 / 10000^(1 / 4)) ≈ 0.995

PE(3, 2) = sin(3 / 10000^(2 / 4)) ≈ 0.007

PE(3, 3) = cos(3 / 10000^(3 / 4)) ≈ 0.998

Positional encoding for 'I':

[0.841, 0.999, 0.002, 1.000]

Positional encoding for 'like':

[0.909, 0.997, 0.005, 0.999]

Positional encoding for 'pizza':

[0.141, 0.995, 0.007, 0.998]

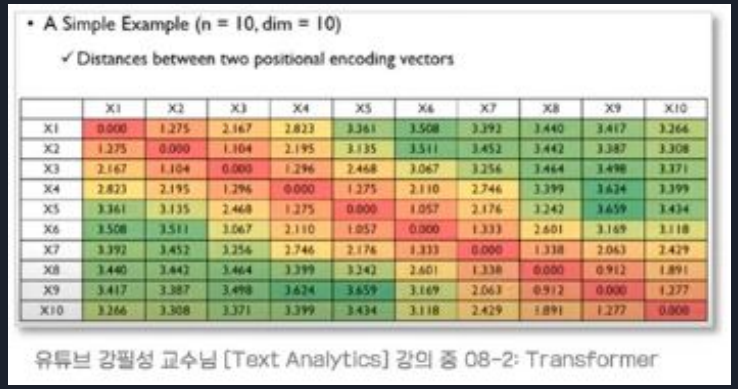
트랜스포머(Transformer) 파헤치기 - 1. Positional Encoding

2.2 Positional Encoding

I: [0.1 + 0.841, 0.2 + 0.999, 0.3 + 0.002, 0.4 + 1.000] = [0.941, 1.199, 0.302, 1.400]

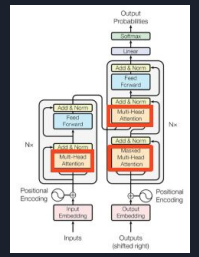
like: [0.5 + 0.909, 0.6 + 0.997, 0.7 + 0.005, 0.8 + 0.999] = [1.409, 1.597, 0.705, 1.799]

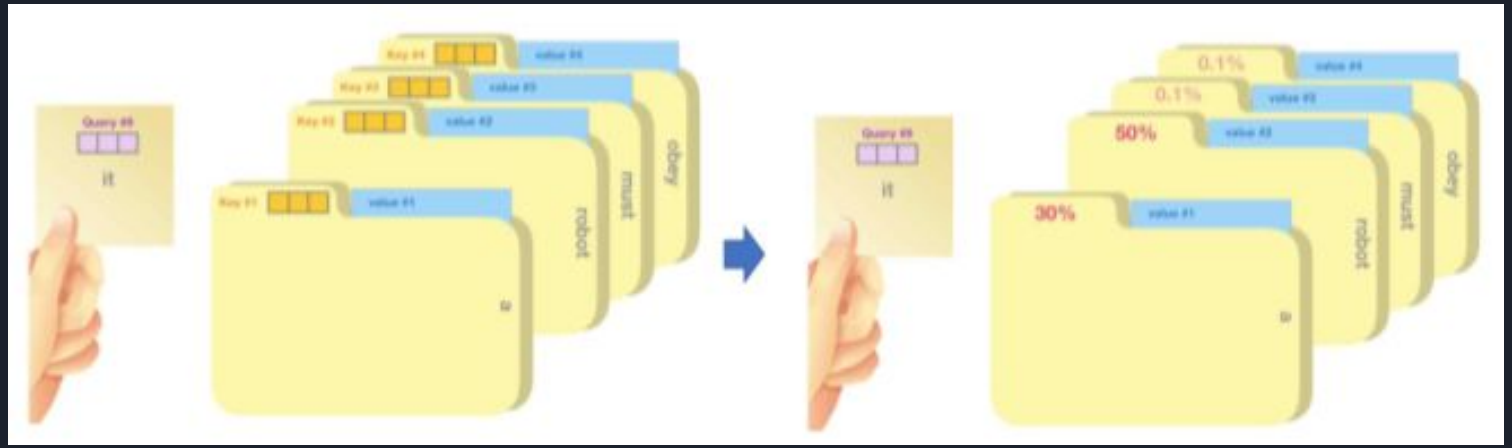
pizza: [0.9 + 0.141, 1.0 + 0.995, 1.1 + 0.007, 1.2 + 0.998] = [1.041, 1.995, 1.107, 2.198]



	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10
X1	0.000	1.275	2.167	2.823	3.361	3.508	3.392	3.440	3.417	3.266
X2	1.275	0.000	1.104	2.195	3.135	3.511	3.452	3.442	3.387	3.308
X3	2.167	1.104	0.000	1.296	2.468	3.067	3.256	3.464	3.498	3.371
X4	2.823	2.195	1.296	0.000	1.275	2.110	2.746	3.399	3.624	3.399
X5	3.361	3.135	2.468	1.275	0.000	1.057	2.176	3.242	3.659	3.434
X6	3.508	3.511	3.067	2.110	1.057	0.000	1.333	2.601	3.169	3.118
X7	3.392	3.452	3.256	2.746	2.176	1.333	0.000	1.338	2.063	2.429
X8	3.440	3.442	3.464	3.399	3.242	2.601	1.338	0.000	0.912	1.891
X9	3.417	3.387	3.498	3.624	3.659	3.169	2.063	0.912	0.000	1.277
X10	3.266	3.308	3.371	3.399	3.434	3.118	2.429	1.891	1.277	0.000

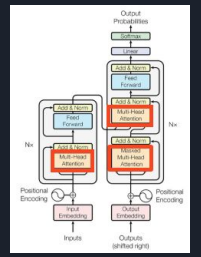
2.3 Self - Attention



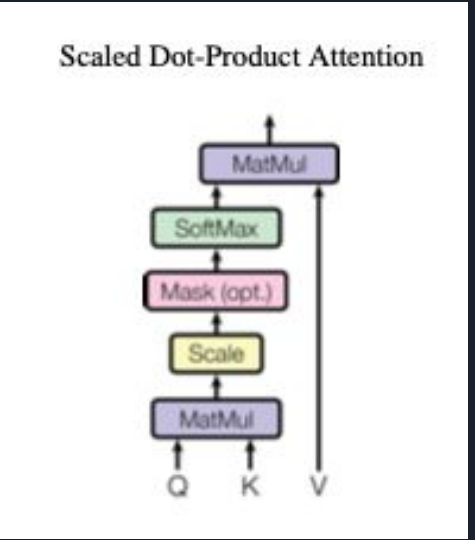


- 그림 자료 : 고려대학교 산업경영공학부 연구실 08-2: Transformer

2.3 Self - Attention

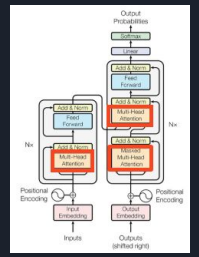


- Attention 을 자기 자신에 대해(self) 수행 한다. 스스로의 연관관계를 파악
- Step1 : Q, K, V vector 생성
- Step2 : 모든 K vector 에 대하여 attention score 를 구한다.
- Step3 : Softmax 로 Attention 분포
- Step4 : V vector 를 가중합 하여 Attention Value를 구한다.



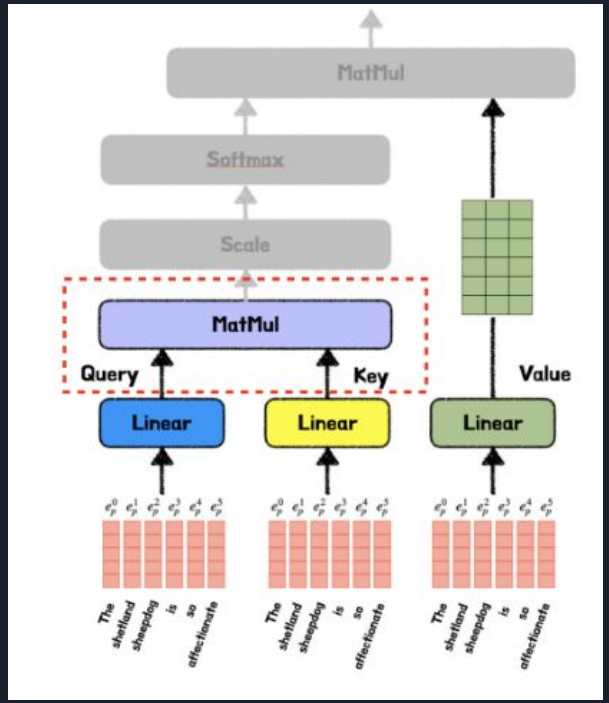
- 그림자료 1
- 그림자료 2
- ratsgo's NLPBOOK 셀프 어텐션 동작 원리

2.3 Self - Attention



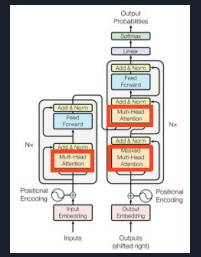
- 셀프 어텐션은 본질적으로 Query, Key, Value가 동일
- Q, K, V : 입력 문장의 모든 단어 벡터들
- Query 와 Key 의 내적을 통해 유사도를 계산한다. 그 결과 값에 Value를 내적하여 Value를 갱신한다.
- 스케일링이 필요한 이유는 과도하게 큰 값을 제거하고 너무 작은 값도 적당하게 바꿈으로써 gradient 전달이 잘 되게 하기 위함.

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



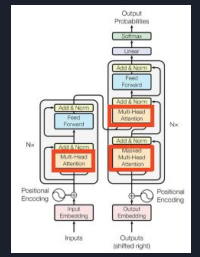
- 딥러닝을 이용한 자연어 처리 입문 16-1 트랜스포머(Transformer)
- 트랜스포머(Transformer) 파헤치기 - 2 Multi-Head Attention

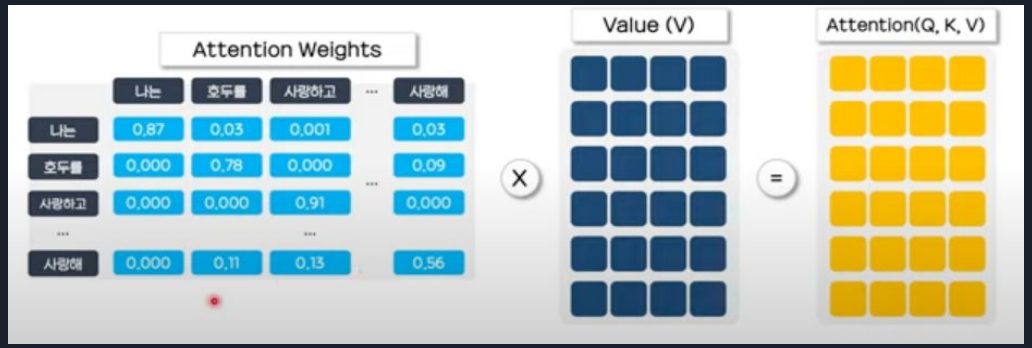
2.3 Self - Attention



- 딥러닝을 이용한 자연어 처리 입문 16-1 트랜스포머(Transformer)
- 트랜스포머(Transformer) 파헤치기 - 2 Multi-Head Attention

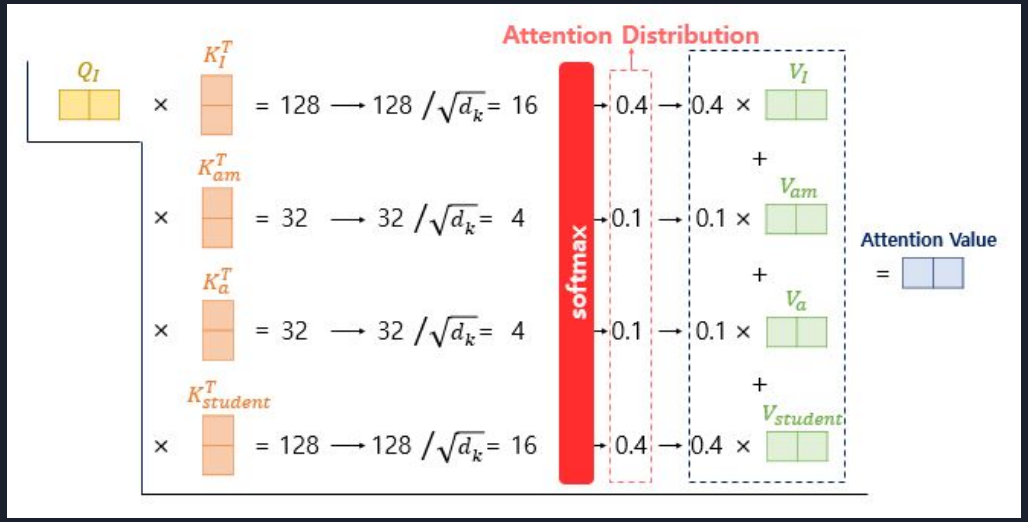
2.3 Self - Attention

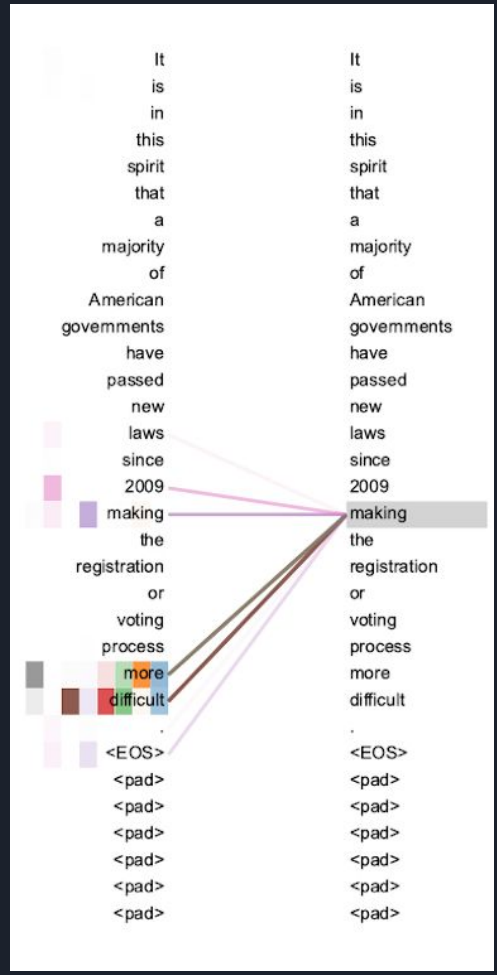


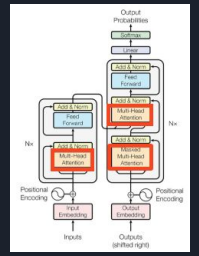


- 딥러닝을 이용한 자연어 처리 입문 16-1 트랜스포머(Transformer)
- 트랜스포머(Transformer) 파헤치기 - 2 Multi-Head Attention

2.3 Self - Attention



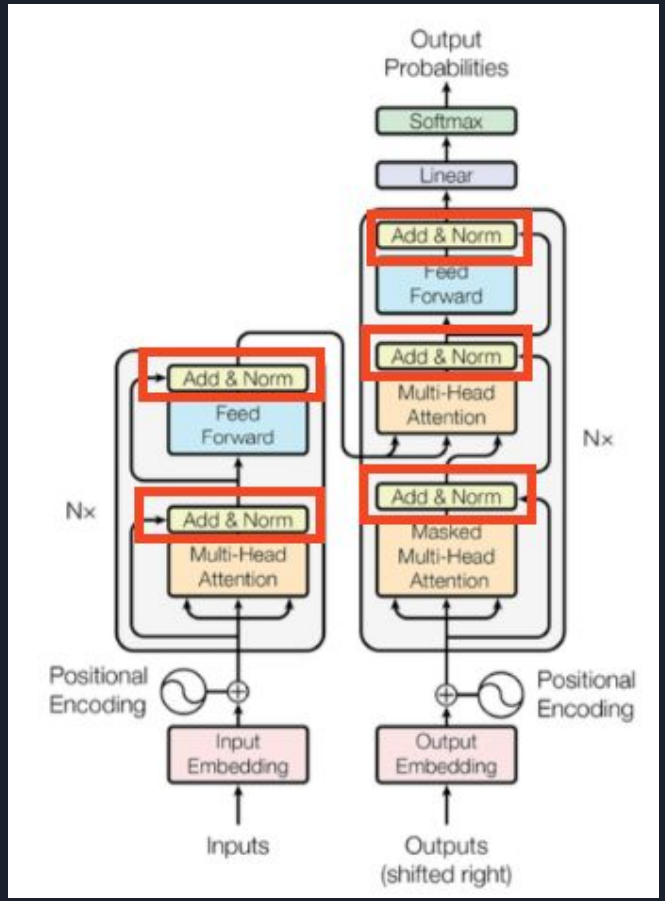




- 그림자료 1
- 그림자료 2
- ratsgo's NLPBOOK 셀프 어텐션 동작 원리

2.4 Add & Norm

- Residual Connection
  - Gradient Vanishing 문제 해결
- Layer Normalization
  - 평균 0, 표준편차 1 로 normalize

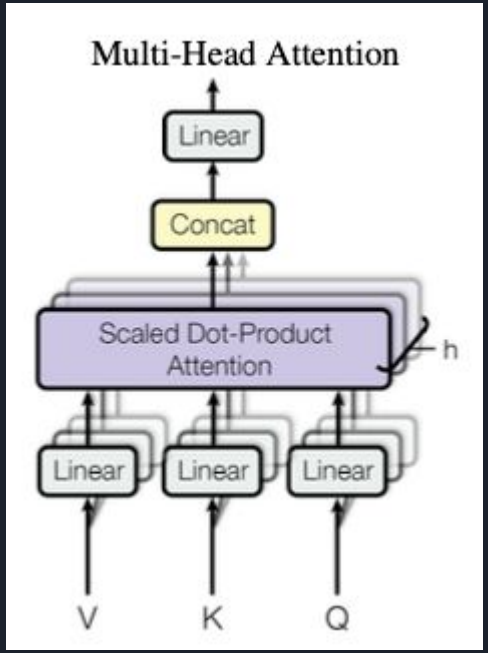


2.5 Multi-head Attention

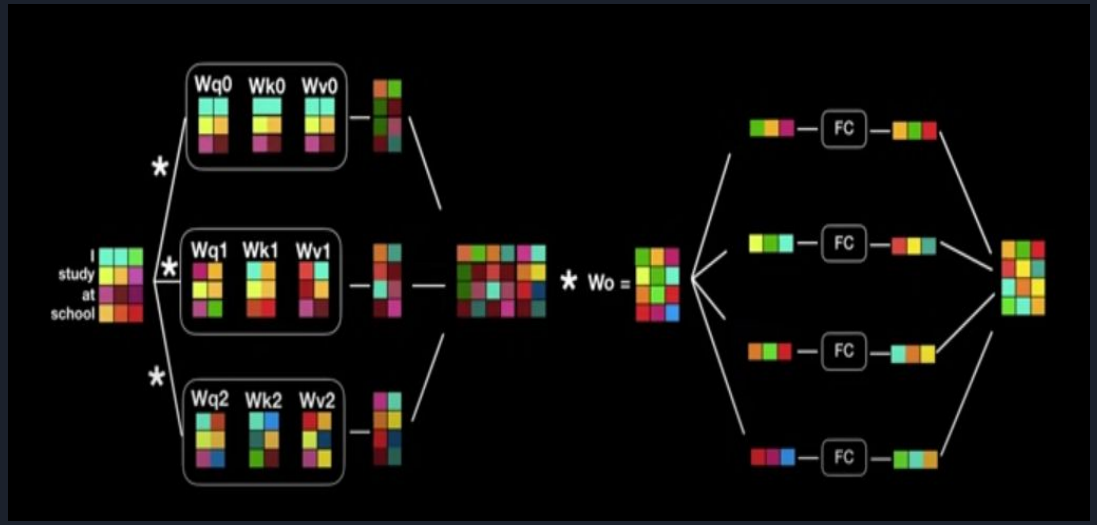
- 병렬로 multi-head를 사용함으로써 여러 부분에 동시에 어텐션을 가할 수가 있어서 모델이 입력 토큰 간의 다양한 유형의 종속성을 포착하고 동시에 모델이 다양한 소스의 정보를 결합할 수 있음

- Wk, Wq, Wv, Wo 는 학습 되는 Parameter

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



2.5 Multi-head Attention



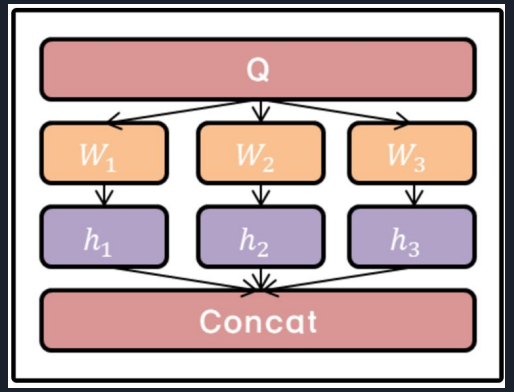
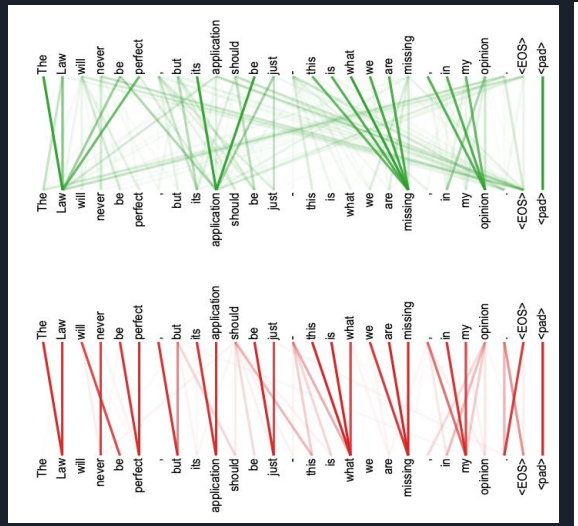
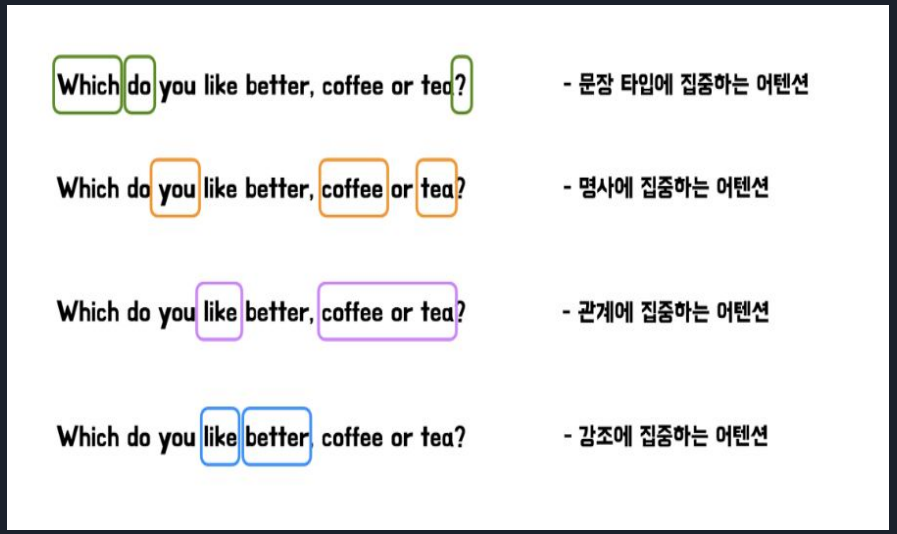


그림 출처 : 허민석. 트랜스포머 (어텐션 이즈 올 유 니드)

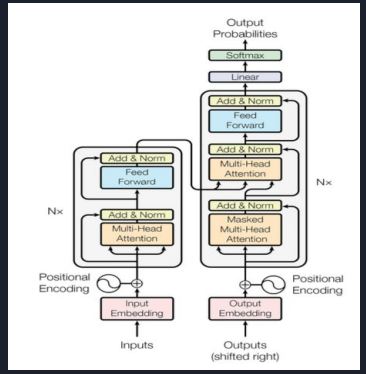
2.5 Multi-head Attention

- 다르게 보면 다르게 보인다.
- 효과적인 학습 가능





2.6 Encoder-Decoder Attention



- Encoder 의 아웃풋은 Key, Value 로써 사용 됨
- Decoder는 Query로 사용된다.
- I am a student 를 프랑스어로 번역하려고 한다면, 추론되는 프랑스어 Query로 영어 단어 Key에 질문을 던지게 되면 프랑스어와 영어의 연관성(어텐션 스코어)를 구한다.

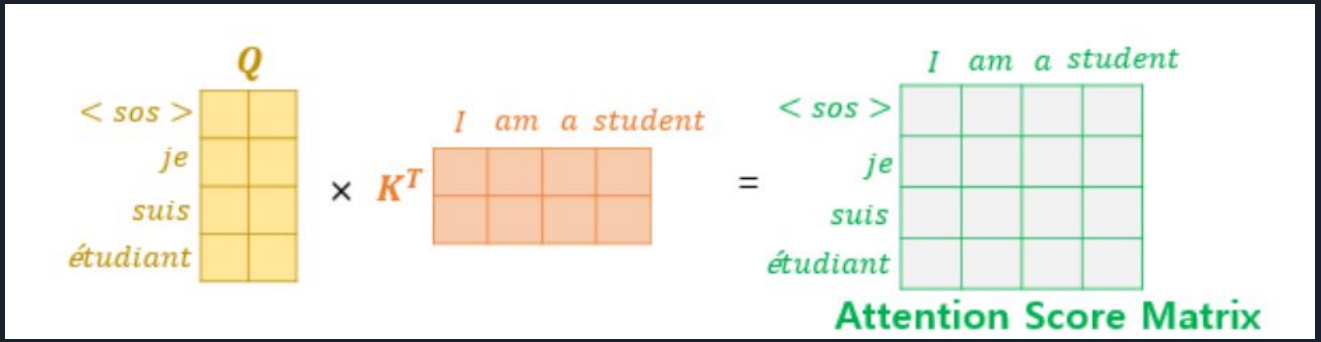
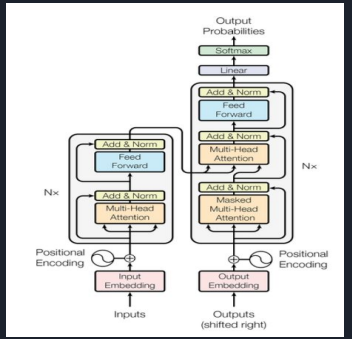
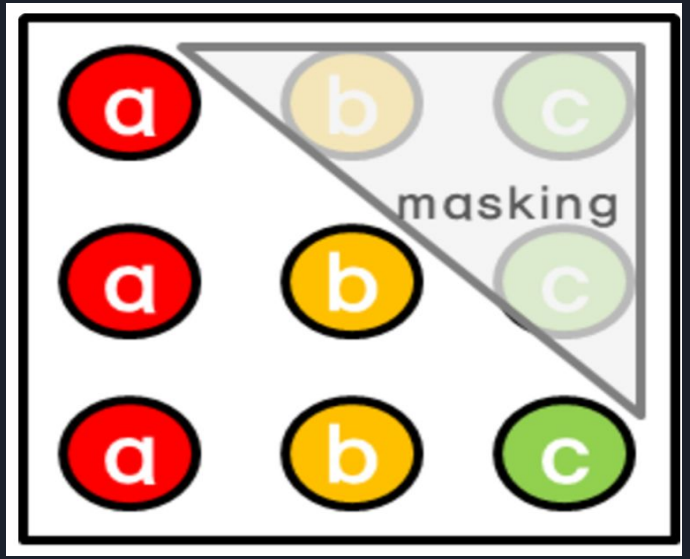


그림 출처

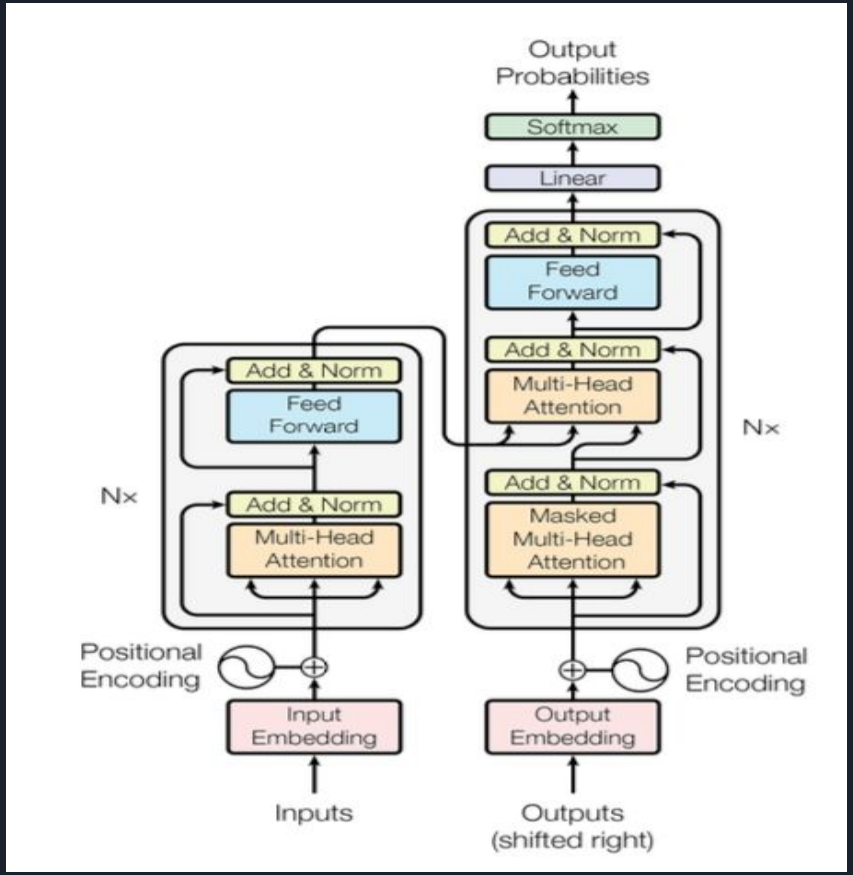
2.7 Masked Multi-head Attention



- 뒤의 정보를 참고하는 것을 방지하기 위해 Attention 가중치에 음의 무한대 값을 넣어줌으로써 소프트 맥스 값에서 0이 되도록 설정



2.8 Transfomer - 전체



3. 실험 결과

- 상대적으로 적은 Training cost 로 우수한 성능
-

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

3. 실험 결과

- (A) -> Single Head, 너무 많은 Head -> 성능 저하
- (B) -> dk 를 줄이면 품질이 저하
- (C) -> 큰 모델이 좋은 성능
- (D) -> 과적합 방지 -> 좋은 성능

	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

4. 결론

- 순환 모델을 배제하고 전적으로 어텐션에만 의존한 최초의 시퀀스 변환 모델 Transfomer
- WMT 2014 영어-독일어 번역과 WMT 2014 영어-프랑스어 번역 태스크에서 트랜스포머는 새로운 SOTA를 성취

5. Q&A 및 논의

📌 **Positional Encoding** 은 실제 위치를 어떻게 보정하는 걸까?

📌 Transformer에 대한 직관적 이해

📌 multi-head attention 은 어떻게 다양한 관점을 반영하는 걸까?

감사합니다

6. Appendix

📌 BLEU Score

$$\text{BLEU} : \#\{w_{gen} \in S_{ref} | w_{gen} \in S_{gen}\} / |S_{gen}|$$

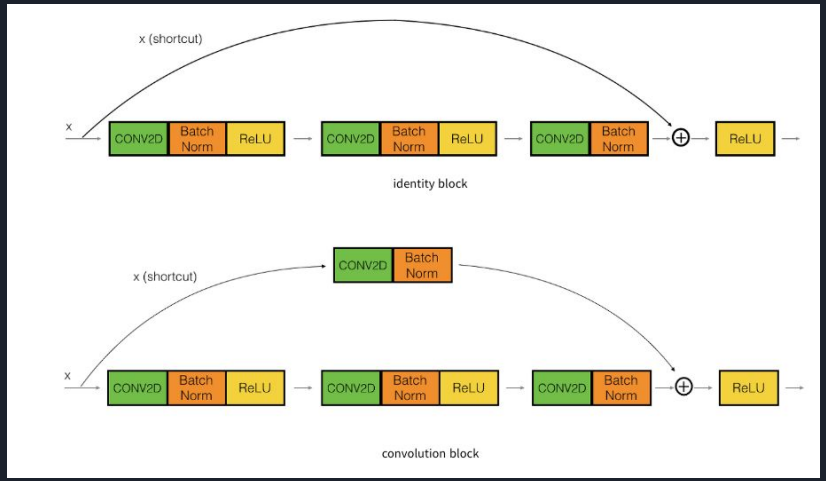
- Generated Sentence의 단어가 Reference Sentence에 포함되는 정도를 말한다.
- 다음과 같은 상황이 있을 때 "i", "was" "by"는 두 문장에 공통으로 들어있다는 걸 알 수 있다.
  - 예측 문장 : I was generated by the model
  - 정답 문장 : I was referenced by human
- 정의된 식에 따라 BLEU Score 는 3/6 이다.

BLEU Score의 소개와 계산하는 방법 (Python)

6. Appendix

📌 Residual Connection 의 기울기 소실 해결 방법

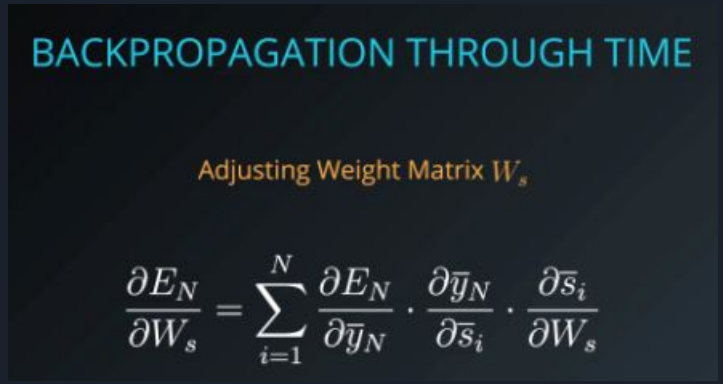
- Residual Connection 이 Vanishing Gradient 를 해결 할 수 있는 이유.



6. Appendix

📌 RNN 에서 Gradient Vanishing 발생 이유

- RNN 에서 Gradient Vanishing 이 발생하는 이유.



6. Appendix

📌 Attention 개념을 제시한 최초의 논문

- 2016, Neural Machine Translation by Jointly Learning to Align and Translate
- https://arxiv.org/abs/1409.0473

6. Appendix

📌 참고 자료

- [Youtube] [딥러닝 기계 번역] Transformer: Attention Is All You Need (꼼꼼한 딥러닝 논문 리뷰와 코드 실습)
- Attention Is All You Need (2017)
- [Youtube] [Paper Review] Attention is All You Need (Transformer)
- The Annotated Transfomer [트랜트포머 Pytorch 코드에 대한 상세 주석 페이지]
- Jay Alammar - The Illustrated Transformer