# Transformer

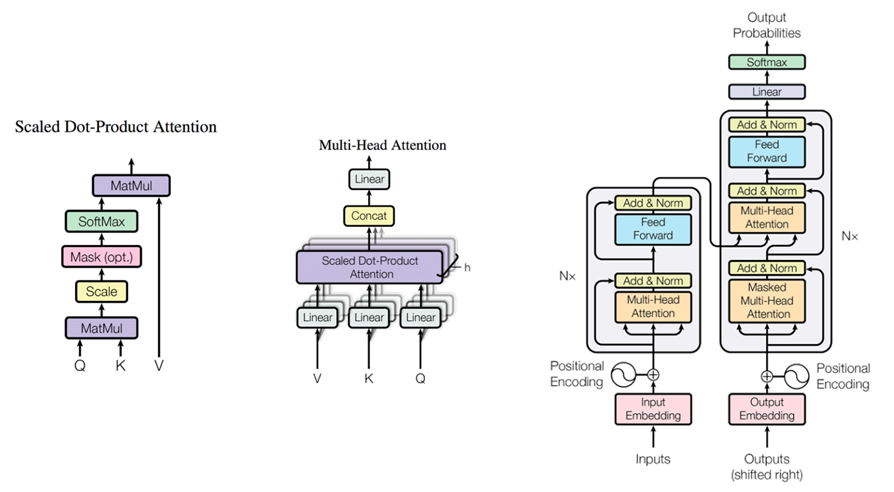
Transformer는 일단 성능과 속도가 모두 기존 모델을 압도한다. 하지만 잘못 사용하면 성능이 떨어지기도 한다. 다음은 기본적인 transformer 구조다.

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명, feedforward is fully connected layer.

Attention은 Query를 generation하는것이다. 좋은 결과를 만들기위해 query를 얻어야 한다.

\* Transformer & Attention



Scale Dot-Product attention에서 scale 부분이 빠지고, attention을 진행한다. 이것을 head의 개수(h)만큼 query, key, value를 만든다.

Attention에서는 Query가 decoder에 매 time-step마다 hidden state를 보낸다. 즉, encoder의 모든 time step의 출력인 key와 value에 query를 날린다.

\* Transformer에는 2가지의 attention이 존재한다.

- Self-Attention: Self-Attention 메커니즘은 입력 시퀀스의 각 요소가 자신과 다른 요소와의 상호작용을 이해하고 관계성을 계산하여 Transformer 모델이 복잡한 문맥적 패턴을 학습할 수 있게 해준다. 이는 Query, Key, Value 벡터 계산, 유사도 점수 계산 및 정규화, 가중치 계산, 그리고 최종 출력 벡터 생성의 과정을 거친다. Decoder의 self-attention에는 ‘Masked Self-Attention’이 존재해 훈련과정에서 미래의 토큰을 보지 못하게 만든다.

- Multi-Head-Attention: 선형변화를 한 후에, Scaled Dot-Product Attention을 계산한다. 이때, Query와 Key의 내적을 계산하고, 그 결과를 Key의 차원 수의 제곱근으로 스케일링한다. Softmax를 통해 attention weight를 얻고, Value를 곱하여 attention을 출력한다. 그리고 Concatenation 및 선형변환을 통해서 Multi-Head-Attention을 최종 출력한다.

\* Transformer Description

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

위 그림에서는 attention이 encoder, decoder, encoder에서 decoder에 날리는 3종류, 6개의 attention이 존재한다. (2개의 encoder와 2개의 decoder)

1. : Encoder의 self-Attention으로 Q, K, V가 같다.
2. : Decoder의 self-Attention으로Q, K, V가 같다. 미래를 보는 것을 방지하는 ‘masked self-attention’이다. 이것은 decoder의 training 과정에만 존재한다.
3. : Encoder-to-Decoder은 Multi-head-attention이다. Decoder의 현재 위치에서 Encoder의 출력에 대한 Query를 수행한다. Decoder의 현재상태를 나타내는 Query를 사용하여 Encoder의 출력에 대한 Key, Value로 attention의 가중치를 계산한다. Seq-to-Seq은 Decoder가 Encoder에 Query를 보내는 것이 아니다.

Multi-Head-Attention과 일반 attention차이는 큰 차이가 없다. Decoder가 query를 Encoder에 날리지 않는 것이다. 위 그림같이 Linear transformation을 각각 해주면 정보를 잘 변화해준다.

\* Transformer Equation

d\_head는 'scaled dot-product attention'에서 scaling을 위해 사용되는 값으로 각 head에서의 차원을 나타낸다. 스케일링을 통해 값의 범위를 조절하고, 학습을 안정화한다. 그냥 attention에서는 이였다. (디코더 1, 인코더 m) 이전에는 각 time step마다 들어갔지만, 지금은 Q가 자기 자신에 대해 모든 time step을 가지고 k에 query를 날린다. 따라서 속도와 메모리가 올라간다.

은 배치 사이즈 문장별 decoder의 time step별 encoder의 time step별 wight

: bs내 각 샘플별 디코더의 time-step별 attention의 결과 vector dimension의 weight.

: bs내 각 time-step의 query를 날렸기 때문에 parallel하게 연산

이전에는 Query를 잘 만들어 key-value를 잘 matching시키고자 했다.

- Multi-head attention은 여러 개의 query를 만들어 다양한 정보를 잘 얻어오고자 했고, Attention 자체로도 정보의 encoding과 decoding이 가능함을 보여주었다(self-attention).

# Encoder Block

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

(시퀀스 길이를 맞추기 위한 패딩마스크가 필요하다)

\* Equation

- Given dataset

X~P(x): p(x)라는 문장의 분포로부터 x를 sampling

Y~p(y|x): x라는 문장이 주어졌을 때 확률분포 y를 samping

- what we want is? – θ

- Equation

Q, K, and V are from previous layer (same value).

Residual connection and Layer Normalizations are used.

는 Multi-Head Attention을 거치지 않고 바로 residual connection으로 들어가는 부분

- Encoder는 self-attention으로 구성되어 있다. 따라서 Q, K, V는 이전 layer의 출력값으로 같은 값이다.

- Seq2Seq의 Attention과 달리, Q도 모든 time-step을 동시에 연산한다. pararrel하게 연산하므로 빠르지만, 메모리를 많이 사용한다.

- Residual connection으로 인해서 깊은 layer 구성이 가능하다 – Big LM의 토대 마련

# Decoder Block

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

Encoder와 마찬가지로 self-attention이지만, 훈련과정에서 미래를 보는 것을 방지하는 ‘masked-self-attention’이 존재한다. <pad>위치에 대한 mask는 모든 time step에 대하여 진행되며 훈련과정에서 미래를 보는 것을 막는다.

Encoder-to-Decoder Attention은 encoder로부터 key, value를 받는다. (시퀀스 길이를 맞추기 위한 패딩마스크)

\* 미래 보는것을 막는 Decoding with Mask(look-ahead-mask) – Decoder self-Attention

Pad가 있는 hidden state에는 attention이 weight가 있으면 안 된다.

-원하는 곳에 를 넣어 0으로 채워준다

도표, 라인, 평면도, 텍스트이(가) 표시된 사진

자동 생성된 설명

\* Equation

- Given dataset

X~P(x): p(x)라는 문장의 분포로부터 x를 sampling

Y~p(y|x): x라는 문장이 주어졌을 때 확률분포 y를 sampling

- what we want is? – θ

- Equation

Decoder self-attention with mask: Teacher forcing할때나 한 번에 들어가 attention. 이식은 training 과정에서 미래를 보는 것 막는 mask다.

-모든 time step에 대하여 mask를 해준다.

: 미니배치의 각 문장 별 decoder의 각 time step에 대해서, decoder 각 time step별 weight

\* Decoder는 2가지의 attention으로 구성된다

- Attention from Encoder (Encoder to Decoder attention):

K와 V는 encoder의 최종 출력 값, Q는 이전 레이어의 출력 값

- self attention with mask:

Q, K, V는 이전 레이어의 출력 값

Attention weight 계산 시, softmax연산 이전에 masking을 통해 음의 무한대를 주어, 미래 time-step을보는 것을 방지(look-ahead-mask)

추론과정에서는 self-attention의 mask는필요 없으나, 모든 layer의 t시점 이전의 모든 time step의 hidden state가 필요하다.

Seq-to-Seq는 마지막 time step에 대한 hidden state만 가지고 있으면 된다.

Transformer는 모든 time step에 대한 hidden state를 다 가지고 있어야 한다.

# Positioning Encoding

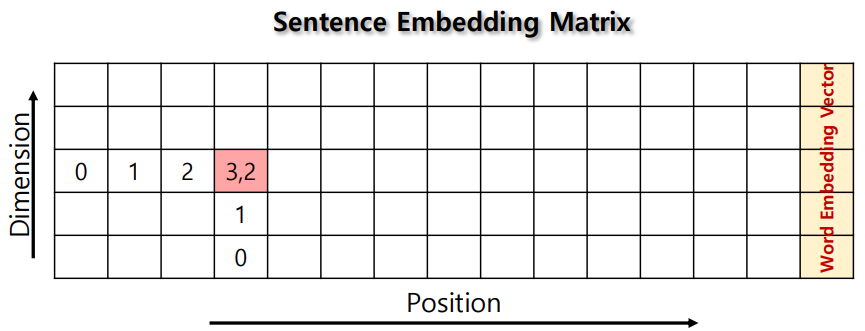
- Transformer는 위치 정보를 스스로 처리하지 않는다.

입력 순서를 바꿔 넣으면 출력도 순서를 바뀐 채 같은 값이 나올 것이다. 마치 FC layer의 입력 feature 순서를 바꿔 학습해도 성능이 똑같은 것과 같다. -> 위치(순서)정보를 따로 인코딩해서 넣어줘야 한다.

자연어 철에선 출력의 위치가 매우 중요하다. RNN의 경우에는 위치정보가 자연스럽게 처리된다(위치 정보를 가지고 있기 때문이다.)

\* Positional Encoding

기존의 word embedding값에 positional encoding값을 더해준다: 학습된 값 + PE



Dimension이 홀수면 cos, 짝수면 sin을 사용한다.

\* vs Positional Embedding

위치 정보도 integer값이므로 embedding layer를 통해 임베딩 할 수도 있다.

BERT와 같은 모델들은 positional encoding대신에 positional embedding을 사용하기도 한다.

\* Transformer는 RNN과 달리, 순서(위치)정보를 encoding해주는 작업이 필요하다. 학습이 아닌 단순 계산 후 encoding이다. RNN은 자연스럽게 수식상 주어진 단어들이나 정보들을 고려해서 encoding되기 때문에 자연스럽게 위치 정보를 가지고 있다. 학습에 의해 달라지는 값이 아니므로, 한 번만 계산해두면 된다.

# Optimizer: Learning rate warm-up and linear decay (Noam)

\*SGD (+ Gradient clipping/ Momentum)

가장 가본적인 방법으로, Learning rate에 따른 성능 변화가 있다. 너무 크면 발산, 작으면 시간이 오래 걸린다.

학습 후반부에 LR decay를 주기도 한다. 예를 들면, epoch10까진 1, 그 이후에 decay ratio를 0.5씩 설정해준다.

\*Adam

Adaptove하게 LR을 조절한다

일부 깊은 네트워크(Transformer)에서는 성능이 낮다. -> 문제는 지금 Transformer세상이다. Transformer에서 사용하면 Adam 성능이 떨어지더라.

* Warp-up and Linear Decay (Noam Decay)

- Heuristic Methods: Control learning rate for Adam with hyper-params

- 학습 초기 불안정한 gradient를 통해 잘못된 momentum을 갖는 것을 방지한다.

Noam decay로 문제를 우회할 수 있는데, 처음에는 LR을 작게 가져 감으로써 불안정한 gradient를 통해 잘못된 momentum을 가져가는 것을 방지하자는 아이디어다. 즉 학습 초기 gradient를 버리거나, 조금만 학습하는 것이다.

Warm-up step은 약 5%로 설정하며 이것은 tuning값이다. 문제는 이것도 hyper-parameter다.

라인, 그래프, 도표, 경사이(가) 표시된 사진

자동 생성된 설명

-결국 trial & error 방식으로 hyper-parameter 튜닝을 해야 한다.

가장 핵심 #warm-up-steps와 #total iterations, 이외에도 다양한 hyper-parameter인 init\_LR, batch\_size들이 존재한다.

심지어 SGD + Gradient Clipping이 더 나은 결과값이 나올수도 있다. (하이퍼 파라미터 결과에 따라서)

\*Rectified Adam

Adam 최적화 알고리즘이 초기 단계에서 불안정하게 작동하는 문제를 지적하고 있습니다. 이는 특히 학습률(learning rate)이 큰 경우에 두드러지는데, 이로 인해 모델의 성능이 저하가발생

따라서 고안된 방법

# Transformer Details Not Described in The Paper

* Transformer 단점

학습이 까다롭다. Bad local optima에 빠지기 쉽기 때문이다. 하지만 paper에는 이것을 언급하지 않는다. 게다가 많은 optimizer를 요구한다. (warm-up size, learning rate, batch\_size) 이에 따라 수많은 transformer model 학습에 대한 논문들이 존재한다.

# Pre-norm layer

이전에는 Noam decay(warm-up and linear decay) 또는 Rectified Adam(RAdam)을 사용했다.

새롭게 제안된 방법으로는 Layer Norm의 위치에 따라서 학습이 수월해진다는 것이다. LN이 gradient를 평탄하게 바꾸는 효과가 있기 때문이다. -> 그냥 layer-norm위치만 바꾸고 Adam을 쓰는 방식이다

\* Layer-norm위치를 Multi-head attention뒤로 옮기고 Adam을 사용하자! 마지막에 Layer-norm까지 추기해줘야 한다.

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

텍스트, 영수증, 폰트, 화이트이(가) 표시된 사진

자동 생성된 설명

\* Pre-Norm방식을 통해 warm-up및 LR튜닝 제거가 가능하다. 하지만 여전히 LR decay가 필요하다. 수렴속도는 빨라지고, 성능은 비슷하다. 8~10 epoch정도 사용하자.

-그밖에도 layer norm을 대체하거나, weight initialization을 활용하여 더 나은 성능을 가질 수 있다.

Hugging face같은경우는 pre layer normalization을 사용한다.

# Warm-up

* Transformer

Attention연산을 통해 정보의 encoding/decoding을 해결한다.

RNN과 달리 위치(순서)정보를 따로 넣어줘야 한다.

- 3가지 attention으로 구성

Self-attention – encoder: pad에 대한 마스크 (시퀀스 길이를 맞추기 위한 패딩마스크)

Self-attention with mask – decoder: training과정에서 미래를 보지 못하는 mask필요

Encoder-to-Decoder – decoder: encoder의 source sentence의 빈칸<pad>를 무시하는 mask필요 (시퀀스 길이를 맞추기 위한 패딩마스크)

Residual connection으로 깊게 쌓을 수 있다.

추후 BERT와 같은 Big LM이 가능해졌다.

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

* Multi-head Attention

\* Attention @ Sequence to Sequence

- Dot-product 연산은 cosine similarity와 매우 유사

- Attention은 query를 잘 만들어내서 key-value로부터 필요한 정보를 얻어내는 과정 – deocder가 encoder에 query를 날린다

- Decoder의 각 time step마다 encoder로부터 attention을 통해 정보를 얻어와 생성 토큰의 품질을 높힌다.

\* Multi-head Attention @ Transformer

- 각 head별로 attention을 수행하여 다양한 정보를 얻어올 수 있다.

- self-attention을 통해 이전 layer의 정보를 encoding/decoding

- decoder가 attention을 통해 encoder의 정보를 얻어온다.

텍스트, 스크린샷, 도표, 폰트이(가) 표시된 사진

자동 생성된 설명

* Optimization

- Layer Normalization과 Residual connection으로 인한 최적화 난이도 증가

- Post-LN Transformer: Big batch size (over 4k),

Noam Decay (Learning rate warm-up and Linear Deacy), Rectified Aadam (RAdam)

- Pre-LN Transformer

Big batch size (over 4k), Adam (+Learning rate decay)

오픈소스를 활용한 경우, 어떤 구조인지 확인하고 사용

텍스트, 도표, 스크린샷, 평면도이(가) 표시된 사진

자동 생성된 설명

* Layer의 위치를 바꾸니까 결과가 좋더라!