Finx_torch

01. Transformer

김남형

Department of Industrial Engineering
Hanyang University

2022-01-09

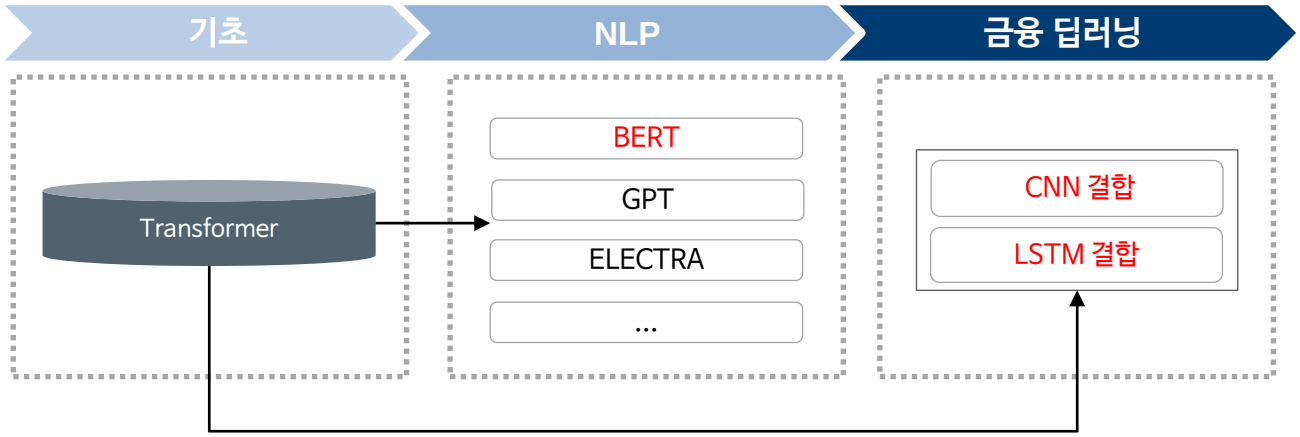
FINANCIAL INNOVATION
& ANALYTICS LAB.

스터디 방향

자연어처리와 금융 딥러닝

- 논문 리스트

순서	Paper
01	Attention is all you need
02	BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding
03	Deep Learning Statistical Arbitrage
04	Accurate Multivariate Stock Movement Prediction via Data-Axis Transformer with Multi-Level Contexts



[Transformer와 그 응용]

자연어처리와 금융 딥러닝

- 스케쥴
  - 2월 28일까지 최대한 4개의 논문을 구현하는 것을 목표
    - » Transformer: 1.5주, BERT: 2.5주
    - » Arbitrage: 2.5주, Stock Prediction: 2주
- 구현 Reference
  - 구현할 때 참고할 reference가 있으면 좋을 것 같음
  - 각자 논문 발제일에 reference로 삼으면 좋을만한 코드(github, colab ..) 조사, 공유
  - 공유된 reference를 기준으로 구현해보면서 질문 공유
- 논문 발제
  - 첫번째 Transformer: 김남형
  - 이후 돌아가며 발제하고자 함.
  - 발제할 때 발표자가 놓친 부분이나 질문이 있으면 언제든지 이야기할 것 !

Transformer

1. Introduction

- Transformer를 쓰는 이유
  - RNN계열은 내재적으로 시퀀스 속성을 지님
    - » 이는 병렬처리를 방해 ▶ 메모리에 제약이 걸림 ▶ 긴 시퀀스에 치명적
  - Attention Mechanism
    - » Input과 output의 거리에 상관없이 dependency를 모델링하게 해줌
    - » 그러나 Transformer 이전 모델(NEURAL MACHINE TRANSLATION
    - » BY JOINTLY LEARNING TO ALIGN AND TRANSLATE)은 recurrent 구조와 결합하여 사용해왔음
  - Transformer 제시
    - » Recurrence 구조 제거. 오직 Attention mechanism에만 의존하여 global dependency를 그린다

2. Background

- CNN 모델과 Self-attention에 대한 설명
  - CNN모델
    - » Operation량이 포지션 사이 거리에 따라 증가함(선형: ConvS2S, logarithm: ByteNet)
    - » 먼 거리의 dependency 배우기 어려움
    - » Transformer에서는 operation량이 상수(c)로 줄어듦
  - 잘 모르겠는 부분
    - » Transformer의 averaging attention-weighted position으로 인해 효과적인 분해(resolution)은 감소하나 Multi-Head Attention으로 대응한다.
    - » Averaging attention-weighted position: $softmax(\frac{QK^T}{\sqrt{d_k}})$가 확률이므로 position(V)가 가중평균 된다는 뜻인 것 같은데… 이로 인해 resolution의 감소가 뜻하는 바는 무엇인지 잘 모르겠음😳
  - Self-attention: 이미 Transformer 이전에 제시된 방법론.
    - » 하나의 시퀀스의 서로 다른 포지션을 relating. ▶ 시퀀스의 representation 계산

3. Model Architecture

- Encoder Decoder

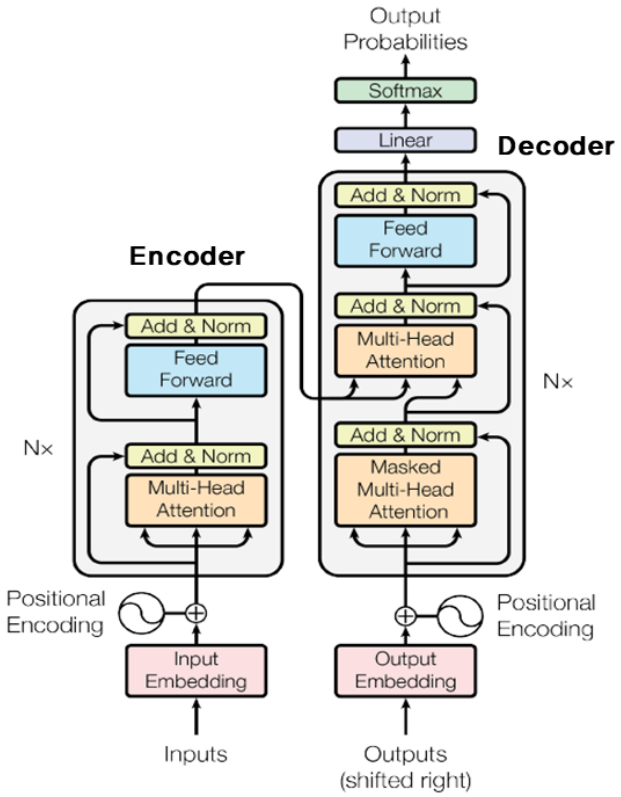
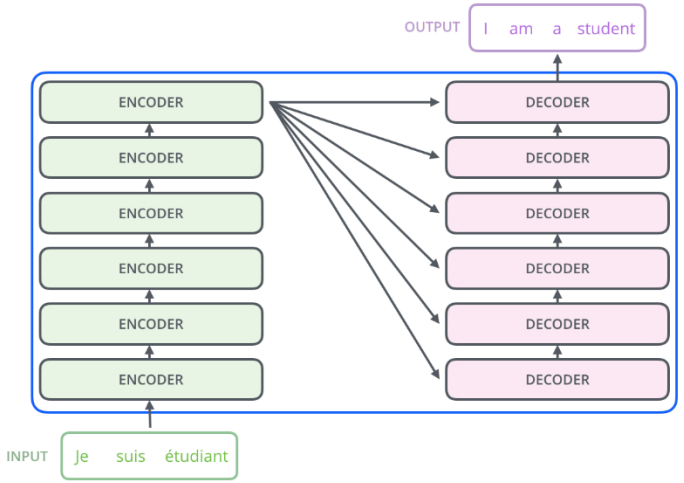


Figure 1: The Transformer - model architecture.

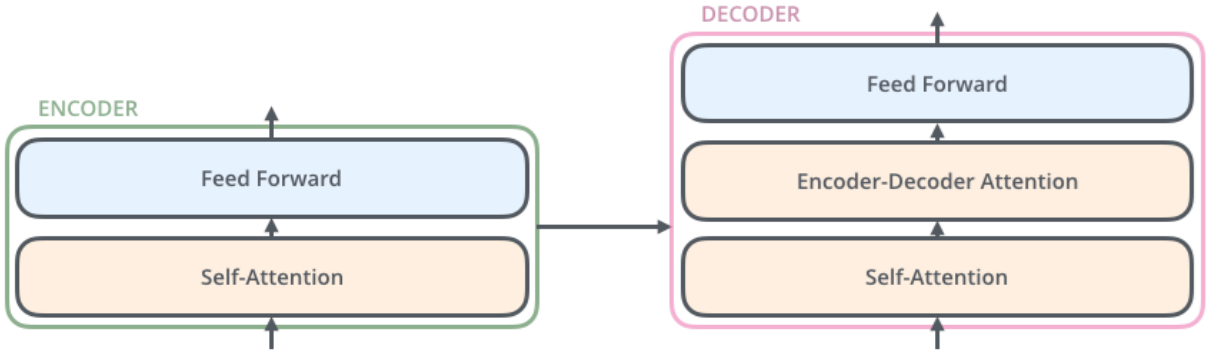


Autoregressive한 성격

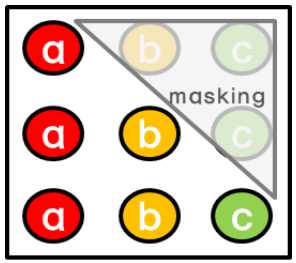
Encoder: $(x_1, \ldots, x_n) \rightarrow (z_1, \ldots, z_n)$
Decoder: $given\ Z, generate(y_1, \ldots, y_n)$

3. Model Architecture

- 3.1 Encoder & Decoder Stacks



- Encoder
  - » 개별 layer는 MH Self-Attention과 position-wise fully connected ffn으로 구성
  - » 여기에 Residual Connection을 추가하고 그 후 Layer 정규화가 뒤따름
  - » 개별 Sublayer 들의 Output: $\mathrm{LayerNorm}(x + Sublayer(x))$
- Decoder
  - » Decoder의 2번째 sublayer는 cross attention.
  - » 또한, subsequent position에 attention이 적용되는 것 막기위해 masking이 도입



3. Model Architecture

- 3.2 Attention

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

  - Output: weighted sum of Value.
  - 개별 value에 할당되는 weight는 compatibility function(query, key)로 계산됨. (이게 어텐션?)

- 3.2.1 Scaled Dot-product Attention
  - 가장 많이 사용되는 어텐션: additive/dot-product
  - d_k가 작을 때는 두 어텐션이 비슷하나 d_k가 커지면 additive가 훨씬 좋았다.
  - Dot-product 개선 ▶ Scaled Dot-product Attention

이름	스코어 함수
dot	$score(s_t, h_i) = s_t^T h_i$
$bahdanau(additive)$	$score(s_t, h_i) = W_a^T \tanh(W_b[s_t : h_i])$
$scaled\ dot$	$score(s_t, h_i) = \frac{s_i^T h_i}{\sqrt{n}}$

3. Model Architecture

- 3.2.1 Scaled Dot-product Attention
  - Softmax의 미분

$$\frac{\partial}{\partial z_j} s_i(\mathbf{z}) = \begin{cases} s_j(\mathbf{z})(1 - s_j(\mathbf{z})), & \text{if } i = j \\ -s_i(\mathbf{z}) s_j(\mathbf{z}), & \text{if } i \neq j \end{cases}$$

» Softmax의 미분계수: 함수의 요소들 끼리 곱으로 결정 ▶ 0또는 1에 아주 가까우면 Gradient 소실

» Q, K는 Layer 정규화를 거치게 되므로 q, k의 요소들은 대략 평균 0, 1을 따른다고 가정할 수 있음

$$q_i, k_i \sim p(0, 1)$$

» 따라서 표준편차 $\sqrt{d_k}$로 나누어 $E[QK]$의 평균과 분산을 0,1에 가깝게 만들고자 함

$$\begin{aligned} \mathbb{E}\left[\mathbf{q} \cdot \mathbf{k}\right] &= \mathbb{E}\left[\sum_{i=1}^{d_k} q_i k_i\right] \\ &= \sum_{i=1}^{d_k} \mathbb{E}\left[q_i k_i\right] \end{aligned}$$

$$\begin{aligned} \mathbb{E}[q_i k_i] &= \sum_{q_i} \sum_{k_i} q_i k_i \, p(q_i, k_i) \\ &= \sum_{q_i} \sum_{k_i} q_i k_i \, p(q_i) p(k_i) \quad \because q_i \text{ and } k_i \text{ are independent} \\ &= \sum_{q_i} q_i \, p(q_i) \sum_{k_i} k_i \, p(k_i) \\ &= \mathbb{E}[q_i] \, \mathbb{E}[k_i] = 0 \quad \because \mathbb{E}[q_i] = \mathbb{E}[k_i] = 0 \end{aligned}$$

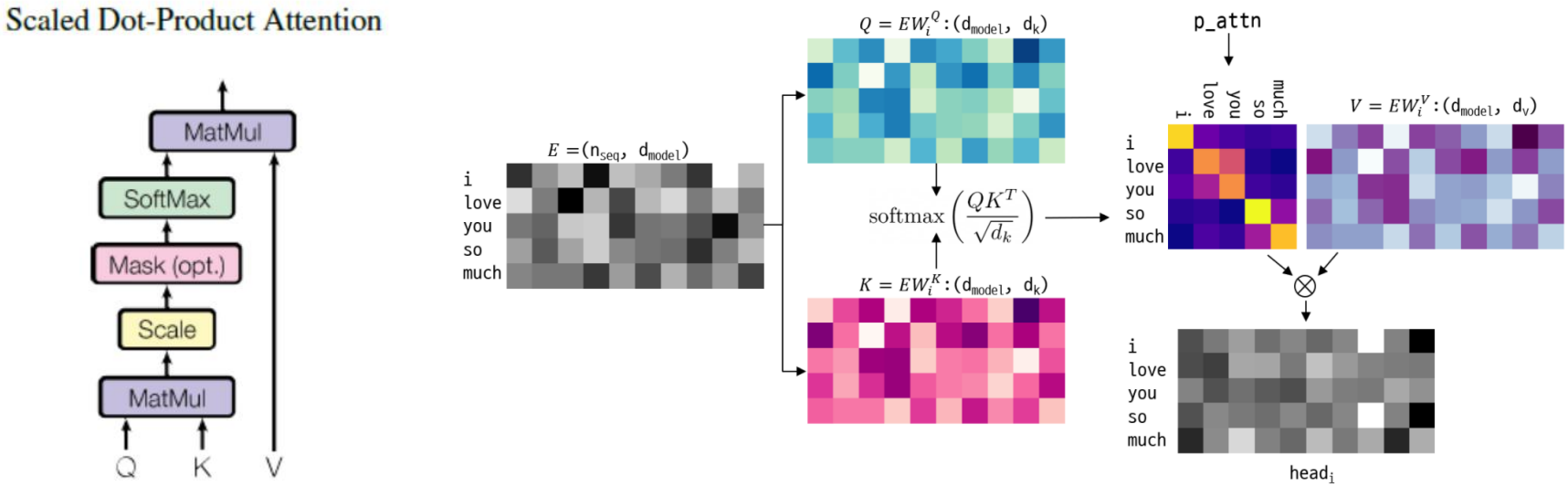
$$\begin{aligned} \mathrm{Var}\left[\mathbf{q} \cdot \mathbf{k}\right] &= \mathrm{Var}\left[\sum_{i=1}^{d_k} q_i k_i\right] \\ &= \sum_{i=1}^{d_k} \mathrm{Var}\left[q_i k_i\right] \end{aligned}$$

$$\begin{aligned} \mathrm{Var}\left[q_i k_i\right] &= \mathbb{E}[q_i^2 k_i^2] - (\mathbb{E}[q_i k_i])^2 \qquad \mathrm{ar}[k_i] = 1 \\ &= \mathbb{E}\left[q_i^2\right] \mathbb{E}\left[k_i^2\right] - \mathbb{E}[q_i]^2 \mathbb{E}[k_i]^2 \\ &= \left(\mathrm{Var}[q_i] + \mathbb{E}[q_i]^2\right) \left(\mathrm{Var}[k_i] + \mathbb{E}[k_i]^2\right) - \mathbb{E}[q_i]^2 \mathbb{E}[k_i]^2 \\ &= \mathrm{Var}[q_i]\mathrm{Var}[k_i] + \mathrm{Var}[q_i]\mathbb{E}[k_i]^2 + \mathbb{E}[q_i]^2 \mathrm{Var}[k_i] + \mathbb{E}[q_i]^2 \mathbb{E}[k_i]^2 - \mathbb{E}[q_i]^2 \mathbb{E}[k_i]^2 \\ &= \mathrm{Var}[q_i]\mathrm{Var}[k_i] + \mathrm{Var}[q_i]\mathbb{E}[k_i]^2 + \mathbb{E}[q_i]^2 \mathrm{Var}[k_i] \;= \mathrm{Var}[q_i]\mathrm{Var}[k_i] = 1 \end{aligned}$$

$$\mathrm{Var}\left[\mathbf{q} \cdot \mathbf{k}\right] = \mathrm{Var}\left[\sum_{i=1}^{d_k} q_i k_i\right] = \sum_{i=1}^{d_k} \mathrm{Var}\left[q_i k_i\right] = \sum_{i=1}^{d_k} \mathrm{Var}[q_i]\mathrm{Var}[k_i] = \sum_{i=1}^{d_k} 1 = d_k$$

3. Model Architecture

- 3.2.1 Scaled Dot-product Attention



3. Model Architecture

- 3.2.2 Multi-Head Attention
  - d_{model}차원으로 한번 어텐션 하지않고 h번 서로 다른 선형 projection ($W \in \mathbb{R}^{d_{model} \times d_k}$) 시킴
    - » 개별 projected q,k,v ($d_k = d_v$ 차원) 에 어텐션 함수를 병렬 처리

$$QW_i^Q = [d_Q \text{x} d_{model}] \text{x} [d_{model} \text{x} d_k] = [d_Q \text{x} d_k]$$

$$KW_i^K = [d_K \text{x} d_{model}] \text{x} [d_{model} \text{x} d_k] = [d_K \text{x} d_k]$$

$$VW_i^V = [d_V \text{x} d_{model}] \text{x} [d_{model} \text{x} d_v] = [d_V \text{x} d_v]$$

$$\downarrow$$

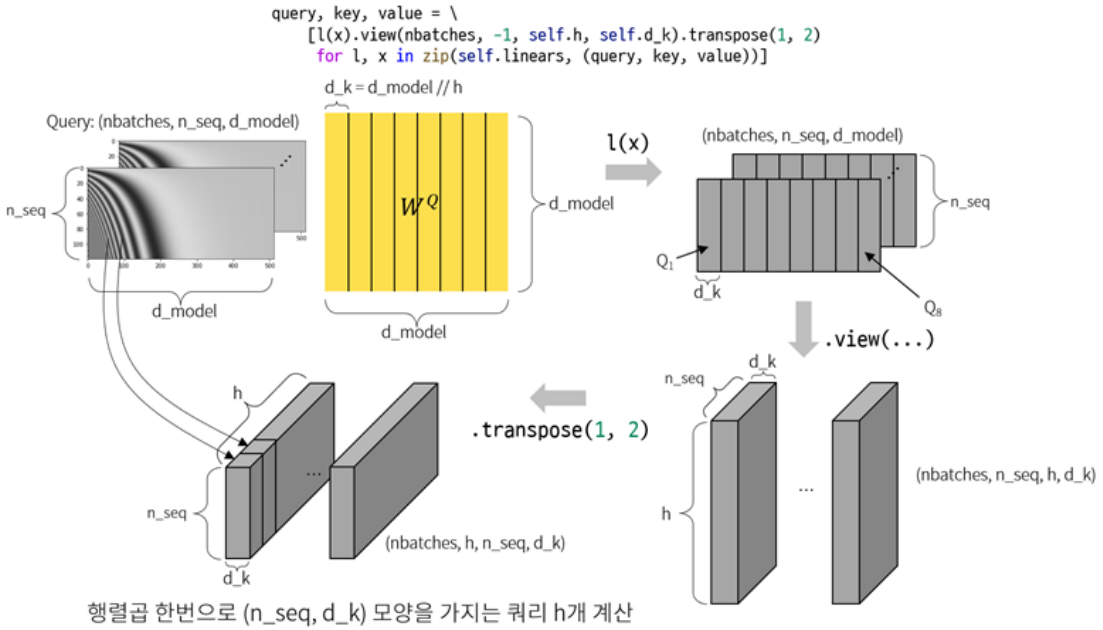
$$Attention(QW_i^Q, KW_i^K, VW_i^V) = [d_V \text{x} d_v]$$

$$\downarrow$$

$$Concat(QW_i^Q, KW_i^K, VW_i^V)W^O = [d_V \text{x} hd_v] \text{x} [hd_v \text{x} d_{model}] = [d_V \text{x} d_{model}]$$

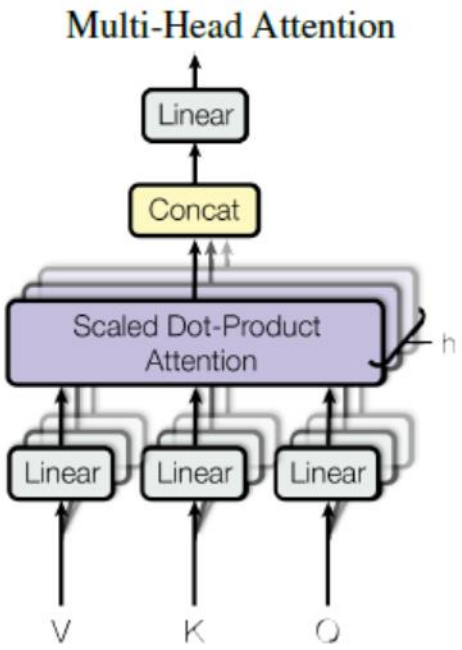
3. Model Architecture

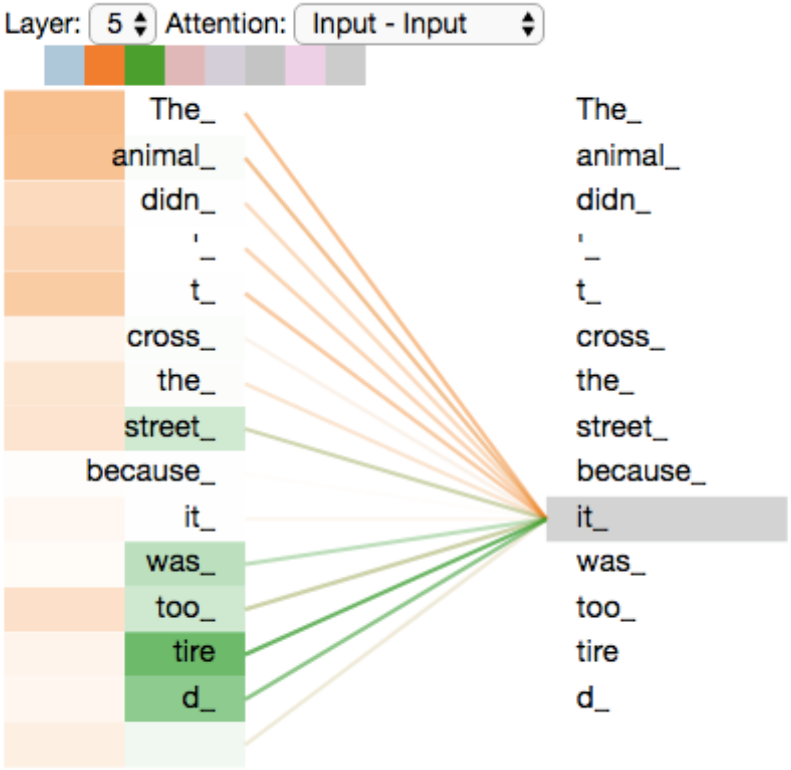
- 3.2.2 Multi-Head Attention
  - 실제 구현에 있어서 개별적으로 여러 번 행렬 곱을 시행 후 Concatenate하지 않음
    - » $(n_{batch}, n_{seq}, d_{model})$ 자체를 $Linear$ 한 뒤 → $(n_{batch},\ head,\ n_{seq},\ d_k)$ 로 변환(Reshape)
    - » $(n_{batch},\ n_{seq},\ head\ *d_k)$
    - » 이와 같이 동작할 경우 여러 개의 weight matrix를 만들 필요가 없어짐(계산, 메모리 효율성)



3. Model Architecture

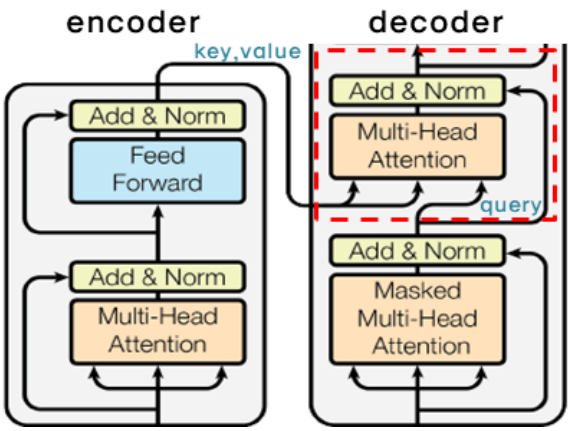
- 3.2.2 Multi-Head Attention

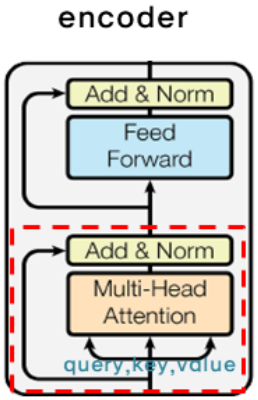


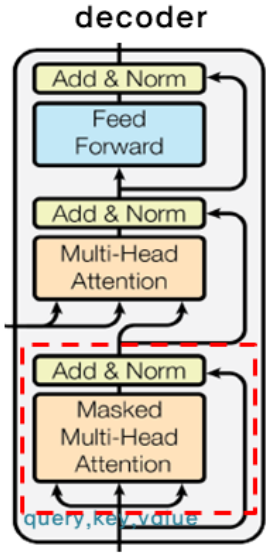


3. Model Architecture

- 3.2.3 Applications of Attention in our Model
  - 총 3개의 방법으로 Attention을 활용함
    - » Encoder-Decoder Attention (Cross Attention): q-이전 디코더층 / k,v – 인코더의 아웃풋
    - » Self-Attention layer in Encoder: q,k,v가 같은 곳에서 도출. 개별 포지션이 모든 포지션에 어텐션
    - » Self-Attention layer in Decoder: AR성격을 위한 masking out

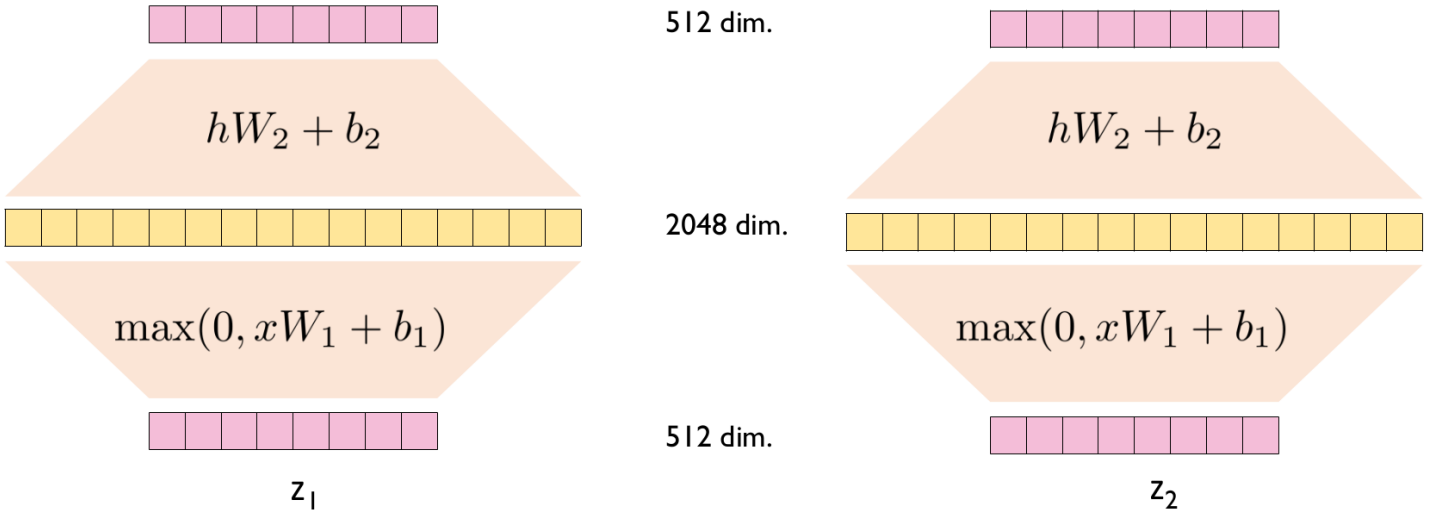






3.3 Position-wise Feed-Forward Network

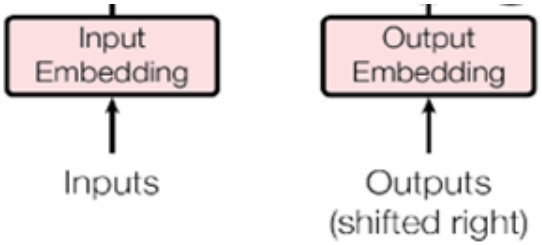
$$FFNN(x) = \max(0, xW_1 + b_1)W_2 + b_2$$



- 잘 모르겠는 점
  - 논문에는 2개의 Convolution(kernel size=1)로 볼 수 있다고 하는데..
    - » Kernel size가 1이고 channel이 layer인 convolution을 두 번 수행한 것으로도 위 과정을 이해할 수 있습니다?

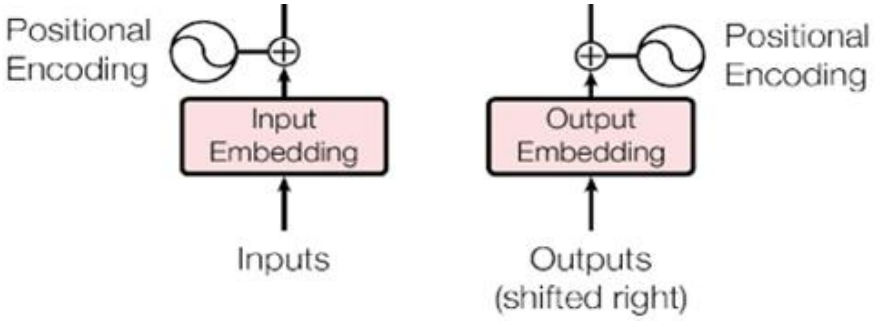
3.4 Embeddings and Softmax

- Embeddings



- $\sqrt{d_{model}}$ 을 임베딩 층에 곱하는 이유? (비공식)
  - » $\sqrt{512} \approx 22$ 정도로 이 값이 임베딩 벡터에 곱해지게 됨
  - » Positional Encoding이 더해지는 것으로 인해 임베딩 결과가 희석되는 것 방지하기 위함

3.5 Positional Encoding



- 포지션 정보
  - RNN과 같이 순차적 구조가 아니기에 단어 간 순서를 반영하기 위해 적용
  - Sinusoid 함수

$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

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

  - » 1. 각 위치마다 유일한 값 출력 – 문장의 총 길이에 무관하게 (Error: 0, 1, ,2 , ⋯ , n)
  - » 2. 길이가 다른 문장의 단어 위치도 상대적 거리가 같으면 같은 값 출력(Error: divided by Seq_N)
  - » 3. 상대 위치의 선형성 확보 (모델 학습 측면): 주기함수적 측면 $PE_{pos+k} = PE_{pos}$

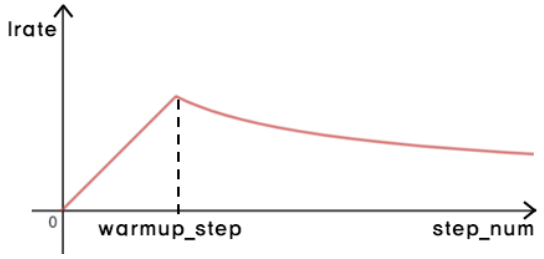
4. Why Self-Attention

- Recurrent, Convolution과 비교했을 때
  - 1. total computational complexity per layer
  - 2. 병렬화 되는 계산량
  - 3. 먼 거리 사이의 dependency

- 차후 연구 언급

5. Training

- 5.1 Training Data and Batching
  - WMT2014
    - » English – German 4.5백만개 문장 pair. 37000 token (BPE)
    - » English – French 3천6백만개 문장 pair. 32000 token (Wordpiece)
    - » 개별 training batch가 대략 25000 src token, 25000 tarket token으로 구성
- 5.2 Hardware and Schedule
  - NVIDIA P100 * 8개
    - » (base) 100,000 step 12시간. (large) 300,000 step 3.5일.
- 5.3 Optimizer
  - Adam
  - Learning rate warm up



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

- » 초기 $step\_num \cdot warmup\_steps^{-1.5} = \min\left(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5}\right)$
- » $step_num$이 증가함에 따라 선형적으로 학습률 증가, $step_{num} = warmup_{steps}$ 일 때 두 항이 같아짐
- » 이후 제곱에 반비례하여 천천히 줄어들게 됨

5.4 Regularization

- Residual Dropout
  - 개별 sublayer의 아웃풋에 먼저 Dropout을 적용한 후 Residual connection
  - Residual connection(Identity Mappings in Deep Residual Networks)

$$x_2 = x_1 + F(x_1, W_1) \qquad x_L = x_l + \sum_{i=1}^{L-1} F(x_i, W_i)$$

$$\frac{\sigma\epsilon}{\sigma x_l} = \frac{\sigma\epsilon}{\sigma x_L}\frac{\sigma x_L}{\sigma x_l} = \frac{\sigma\epsilon}{\sigma x_L}(1 + \frac{\sigma}{\sigma x_l}\sum_{i=1}^{L-1} F(x_i, W_i))$$

  - Dropout(Dropout: a simple way to prevent neural networks from overfitting)
- Label Smoothing(Rethinking the inception architecture for computer vision)
  - Ex. 정답 1 오답 0의 확률 분포를 정답 0.8 오답 0.2로 고루 분배
  - 정답 분포를 '신뢰도' 정도로 해석할 수 있게 하는 방식 $q'(k|x) = (1-\epsilon)\delta_{k,y} + \epsilon u(k)$
- Layer Normalization(Layer Normalization)
  - 같은 layer에 있는 모든 hidden unit이 동일한 평균μ, 분산σ을 공유
    - » 배치 정규화 :

```
mean = x.mean(axis=0)
var = x.var(axis=0, unbiased=False)
```

    - » 레이어 정규화:

```
mean = x.mean(axis=-1).reshape(-1,1,1)
var = x.var(axis=-1, unbiased=False).reshape(-1,1,1)
```

끝

참고 자료

- https://jalammar.github.io
- https://pozalabs.github.io/transformer/
- https://colab.research.google.com/github/metamath1/ml-simple-works/blob/master/transformer/annotated_transformer.ipynb
- https://github.com/pilsung-kang/text-analytics
- https://yngie-c.github.io/nlp/2020/07/01/nlp_transformer/
- https://wikidocs.net/book/2155

FINANCIAL INNOVATION
& ANALYTICS LAB.

Q&A