



Transformer

진행자 : 멘토 현시은

Transformer

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

구글에서 공개한 "Attention is All You Need"라는 논문에서 처음으로 공개된 구조

- Attention을 기반으로 한다.
- 순환구조 때문에 시간이 오래 걸리는 RNN, seq2seq의 단점을 보완
- BERT, GPT-3 등의 언어 모형에서 사용
- 컴퓨터 비전 등 다른 분야에서도 강력한 도구로 사용 중

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

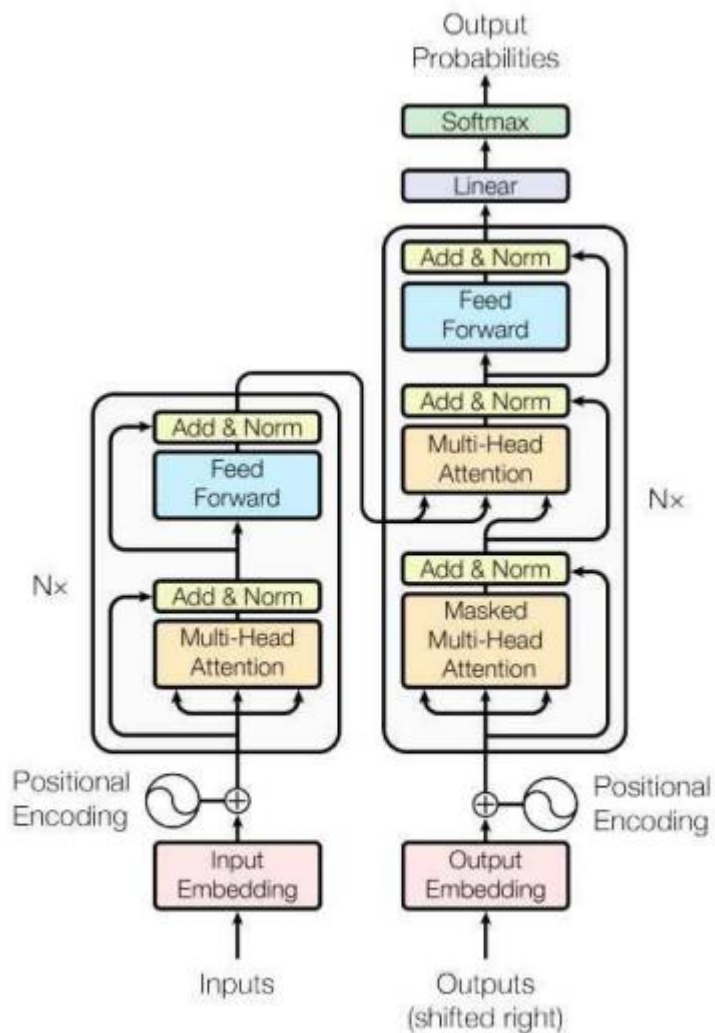


Figure 1: The Transformer - model architecture.

Transformer를 이루고 있는 구성 요소:

- Multihead Attention
- Add & Norm
- Feed Forward
- Masked Multihead Attention
- Positional Encoding
- Linear
- Softmax

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

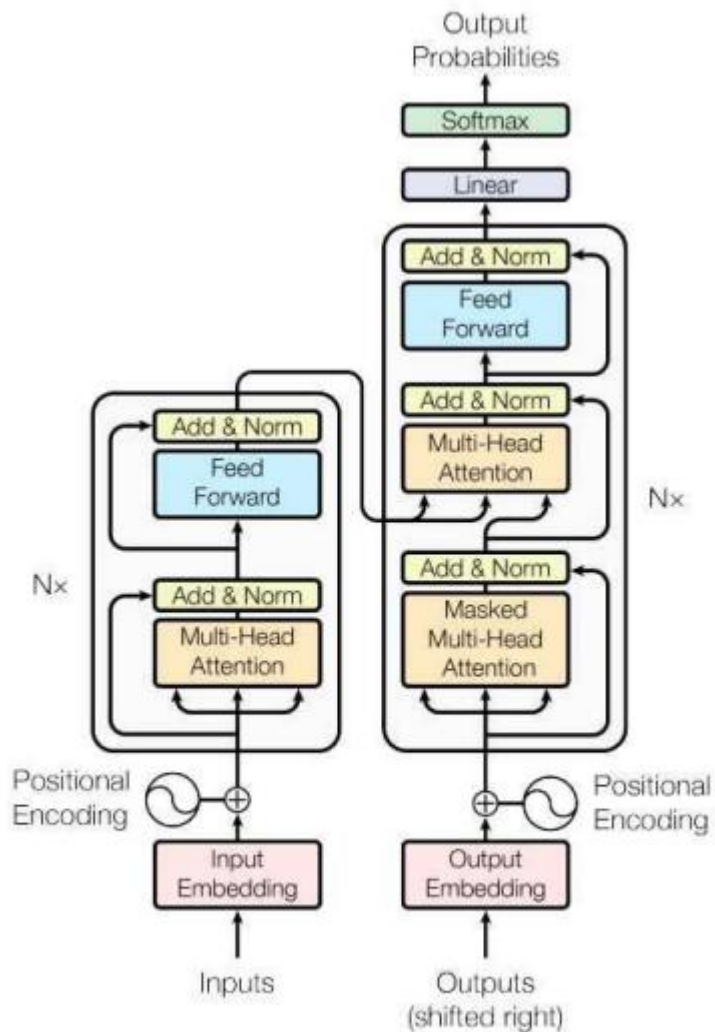


Figure 1: The Transformer - model architecture.

<Transformer는 크게 인코더와 디코더로 나뉜다.>

인코더

-Multihead Attention과 Feed Forward로 이루어진 층이 N 번 반복

디코더

-Masked Multihead Attention과 Multihead Attention, Feed Forward로 이루어진 층이 N 번 반복

-디코더를 통과하여 나온 결과는 차원을 맞추기 위해 Linear(FNN 신경망)을 한번 통과 후 Softmax를 통과

seq2seq

Attention

- Transformer

Q&A Session

공지사항

Transformer

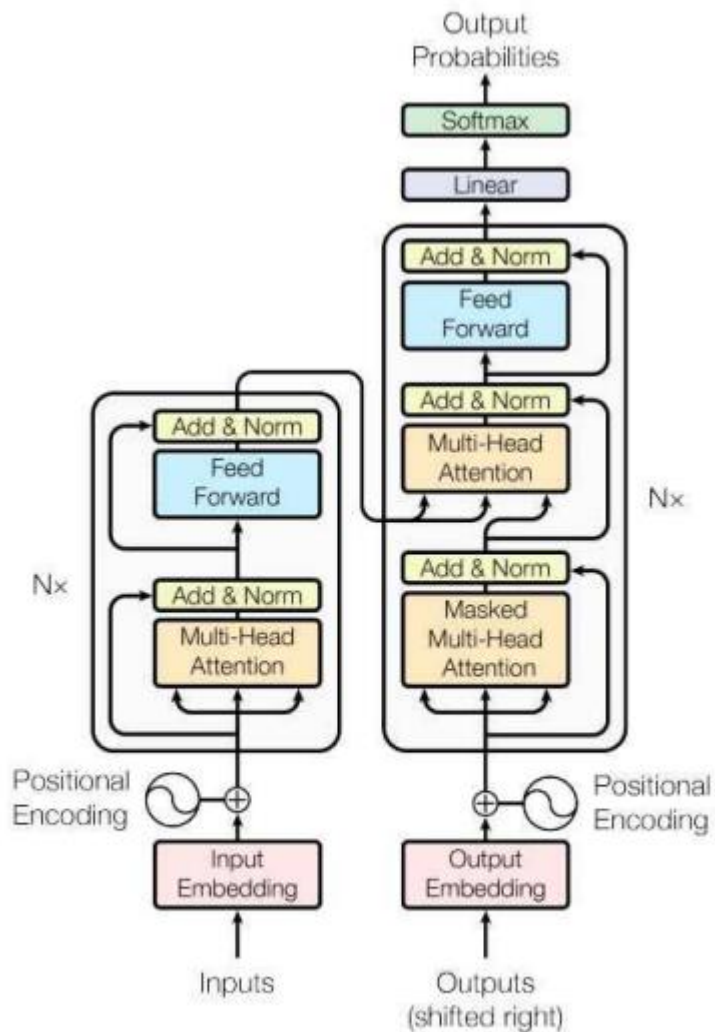


Figure 1: The Transformer - model architecture.

<Transformer는 크게 인코더와 디코더로 나뉜다.>

-인코더와 디코더에 입력될 때에는 Positional Encoding이라는 것을 더한다. 이로 인해 순환구조가 아님에도 위치를 인지할 수 있다.

-Add & Norm은 Residual Connection과 Layer Normalization을 의미함.

seq2seq

Attention

- **Transformer**

Q&A Session

공지사항

Transformer

Transformer를 이해하기 위해 먼저 알아야 할 것들

- Batch Normalization
- Layer Normalization
- Residual Connection

seq2seq

Attention

• Transformer

Q&A Session

공지사항

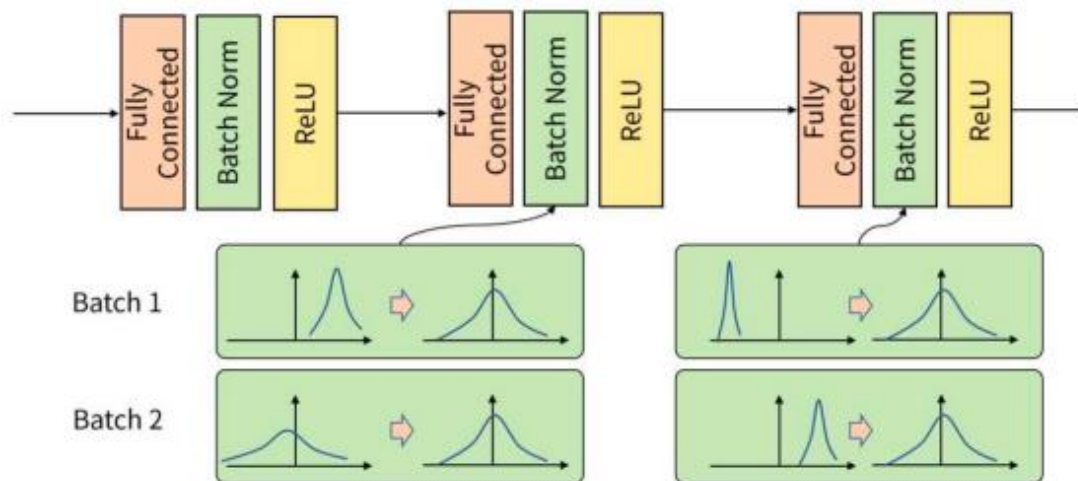
Batch Normalization

입력 데이터를 정규화하는 것은 학습 속도 증가 및 정확한 학습에 필수적이다.

그런데, 처음 입력을 정규화했다고 해서 dense, convolution, ReLU, Softmax 등 수많은 Layer를 거치고 나서도 정규화가 유지될 지는 확실하지 않다.

- 실제로, 층이 깊어질수록 입력 데이터를 정규화한 효과는 없어진다.

그렇다면 층을 거칠 때마다 정규화를 진행한다면?



seq2seq

Attention

• Transformer

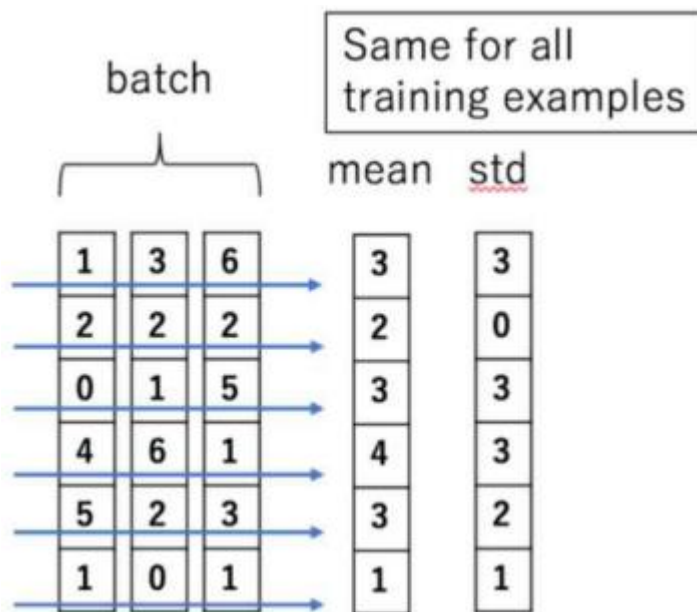
Q&A Session

공지사항

Batch Normalization

Batch Normalization은 Layer를 통과한 이후에 정규화를 적용해주어서 모든 층이 정규화된 데이터를 입력으로 받을 수 있게 해준다.

-테스트 시에는 training 데이터에서 학습한 평균과 표준편차 값을 사용하여 정규화한다.



단점

- Batch의 크기가 어느 정도 커져야 한다.
- 테스트용 데이터의 차원이 트레이닝 데이터의 차원보다 커지면 사용이 불가능 (이로 인해 NLP에서는 사용이 부적절하다.)

seq2seq

Attention

• Transformer

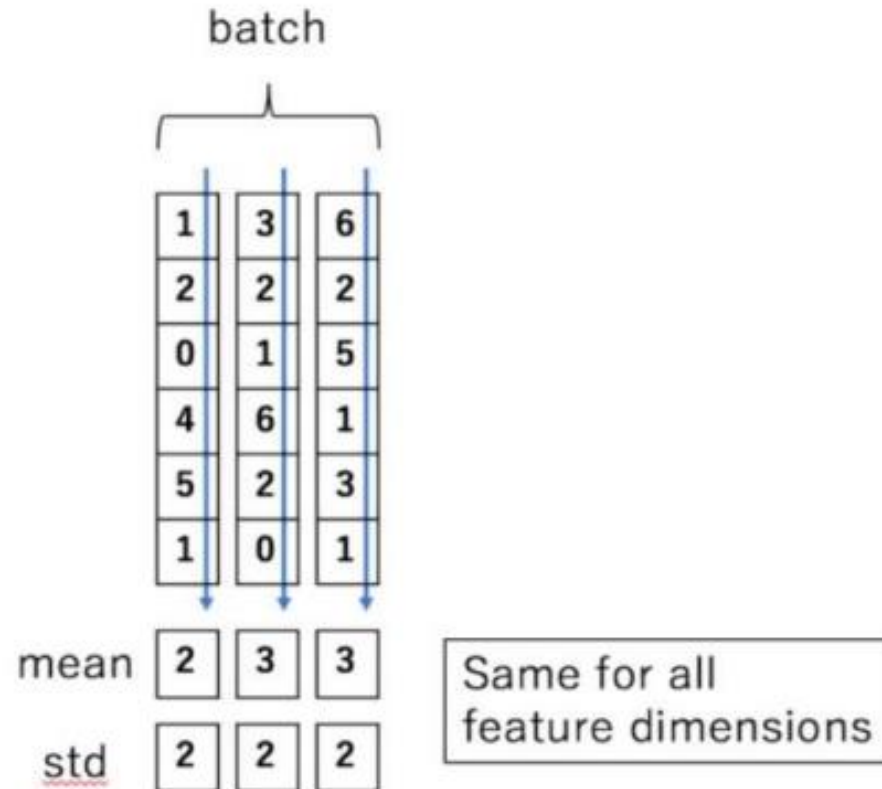
Q&A Session

공지사항

Layer Normalization

데이터셋 내의 데이터별로 모든 차원에 대한 평균과 표준편차를 구한다.

- 데이터마다 차원의 개수가 바뀌는 것에 무관하게 적용 가능하다.
- NLP에서 사용하기 적합하다.
- 시퀀스가 길어져도 정교화를 적용할 수 있다.



seq2seq

Attention

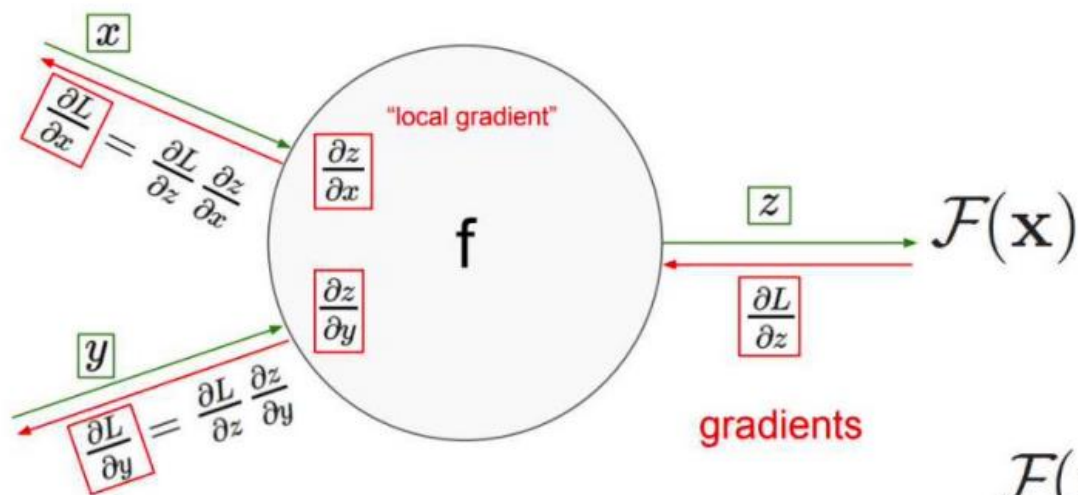
• Transformer

Q&A Session

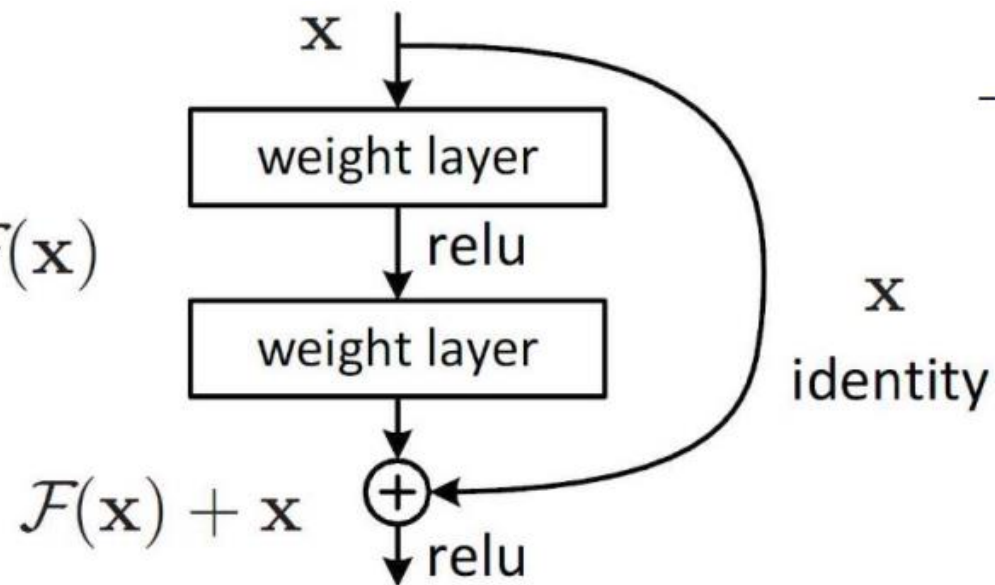
공지사항

Residual Connection

Gradient Vanishing



Residual Connection



- $H(x) = F(x) + x$
- $F(x)$ 의 결과에 대한 출력이 0에 가까운 값이라도 H 의 x 에 대한 기울기는 1 근처로 보장됨

seq2seq

Attention

- **Transformer**

Q&A Session

공지사항

Transformer

· Scaled Dot Product Attention

Transformer의 핵심은 '디코더의 다음 hidden state를 어떻게 정확하게, 효율적으로 산출할 것인가?' 이다.

총 3가지의 변수를 이용하여 다음 hidden state를 구한다.

- **Q(query)**: 직전 디코더의 hidden state => (t, d)의 shape를 가진다
- **K(key)**: 인코더의 전체 hidden state => (T, d)의 shape를 가진다.
- **V(value)**: 인코더의 전체 hidden state => (T, d)의 shape를 가진다.

총 Attention Score를 Q와 K의 내적(dot product)을 통해 계산한다.

seq2seq

Attention

- Transformer

Q&A Session

공지사항

Transformer

- Scaled Dot Product Attention

$$\begin{aligned} Q = \begin{bmatrix} \vec{q}_1 \\ \vec{q}_2 \\ \vdots \\ \vec{q}_t \end{bmatrix} &\longrightarrow E = QK^T = \begin{bmatrix} \vec{q}_1 \\ \vec{q}_2 \\ \vdots \\ \vec{q}_t \end{bmatrix} \begin{bmatrix} \vec{k}_1 & \vec{k}_2 & \dots & \vec{k}_T \end{bmatrix} \\ &= \begin{bmatrix} \vec{q}_1 * \vec{k}_1 & \vec{q}_1 * \vec{k}_2 & \dots & \vec{q}_1 * \vec{k}_T \\ \vec{q}_2 * \vec{k}_1 & \vec{q}_2 * \vec{k}_2 & \dots & \vec{q}_2 * \vec{k}_T \\ \vdots & \vdots & & \vdots \\ \vec{q}_t * \vec{k}_1 & \vec{q}_t * \vec{k}_2 & \dots & \vec{q}_t * \vec{k}_T \end{bmatrix} = \begin{bmatrix} e_{1,1} & e_{1,2} & \dots & e_{1,T} \\ e_{2,1} & e_{2,2} & \dots & e_{2,T} \\ \vdots & \vdots & & \vdots \\ e_{t,1} & e_{t,2} & \dots & e_{t,T} \end{bmatrix} \end{aligned}$$

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

• Scaled Dot Product Attention

Attention Score에 Softmax를 취해 Attention Weight를 계산해주려 한다. 그 전에, 내적값이 너무 커져서 0에 가까운 값이 되는 것을 방지하기 위해 attention score를 \sqrt{d} 로 나눠준다.

$$E' = \frac{1}{\sqrt{d}} E$$

$$a = \text{softmax}(E') = \begin{bmatrix} \text{softmax}(e'_{1,1}) & e'_{1,2} & \dots & e'_{1,T} \\ \text{softmax}(e'_{2,1}) & e'_{2,2} & \dots & e'_{2,T} \\ \vdots & \vdots & \ddots & \vdots \\ \text{softmax}(e'_{t,1}) & e'_{t,2} & \dots & e'_{t,T} \end{bmatrix} \xrightarrow{\text{blue arrow}} C = aV = \begin{bmatrix} a_{1,1} & a_{1,2} & \dots & a_{1,T} \\ a_{2,1} & a_{2,2} & \dots & a_{2,T} \\ \vdots & \vdots & \ddots & \vdots \\ a_{t,1} & a_{t,2} & \dots & a_{t,T} \end{bmatrix} \begin{bmatrix} \vec{v}_1 \\ \vec{v}_2 \\ \vdots \\ \vec{v}_T \end{bmatrix}$$
$$= \begin{bmatrix} \sum_{i=1}^T a_{1,i}(\vec{v}_i)_1 & \sum_{i=1}^T a_{1,i}(\vec{v}_i)_2 & \dots & \sum_{i=1}^T a_{1,i}(\vec{v}_i)_d \\ \sum_{i=1}^T a_{2,i}(\vec{v}_i)_1 & \sum_{i=1}^T a_{2,i}(\vec{v}_i)_2 & \dots & \sum_{i=1}^T a_{2,i}(\vec{v}_i)_d \\ \vdots & \vdots & \ddots & \vdots \\ \sum_{i=1}^T a_{t,i}(\vec{v}_i)_1 & \sum_{i=1}^T a_{t,i}(\vec{v}_i)_2 & \dots & \sum_{i=1}^T a_{t,i}(\vec{v}_i)_d \end{bmatrix} = \begin{bmatrix} \vec{c}_1 \\ \vec{c}_2 \\ \vdots \\ \vec{c}_t \end{bmatrix}$$

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

seq2seq

Attention

- **Transformer**

Q&A Session

공지사항

Transformer

- **Multihead Attention**

- Scaled Dot Product Attention의 경우 연산 과정이 직렬적으로 이어져 있기 때문에 병렬화가 불가능하다. 이로 인해 시간이 오래 걸린다.
- 이를 해결한 것이 Multihead Attention
- 하나의 Attention 연산을 h 개로 쪼개어 동시에 진행하는 방법
- 앞에서 살펴본 Attention(Q, K, V) 함수 하나하나를 attention head라고 부른다.
- 각 attention head마다 다른 feature가 학습되도록 한다.
- 각각의 attention head를 병렬적으로 연산한다.

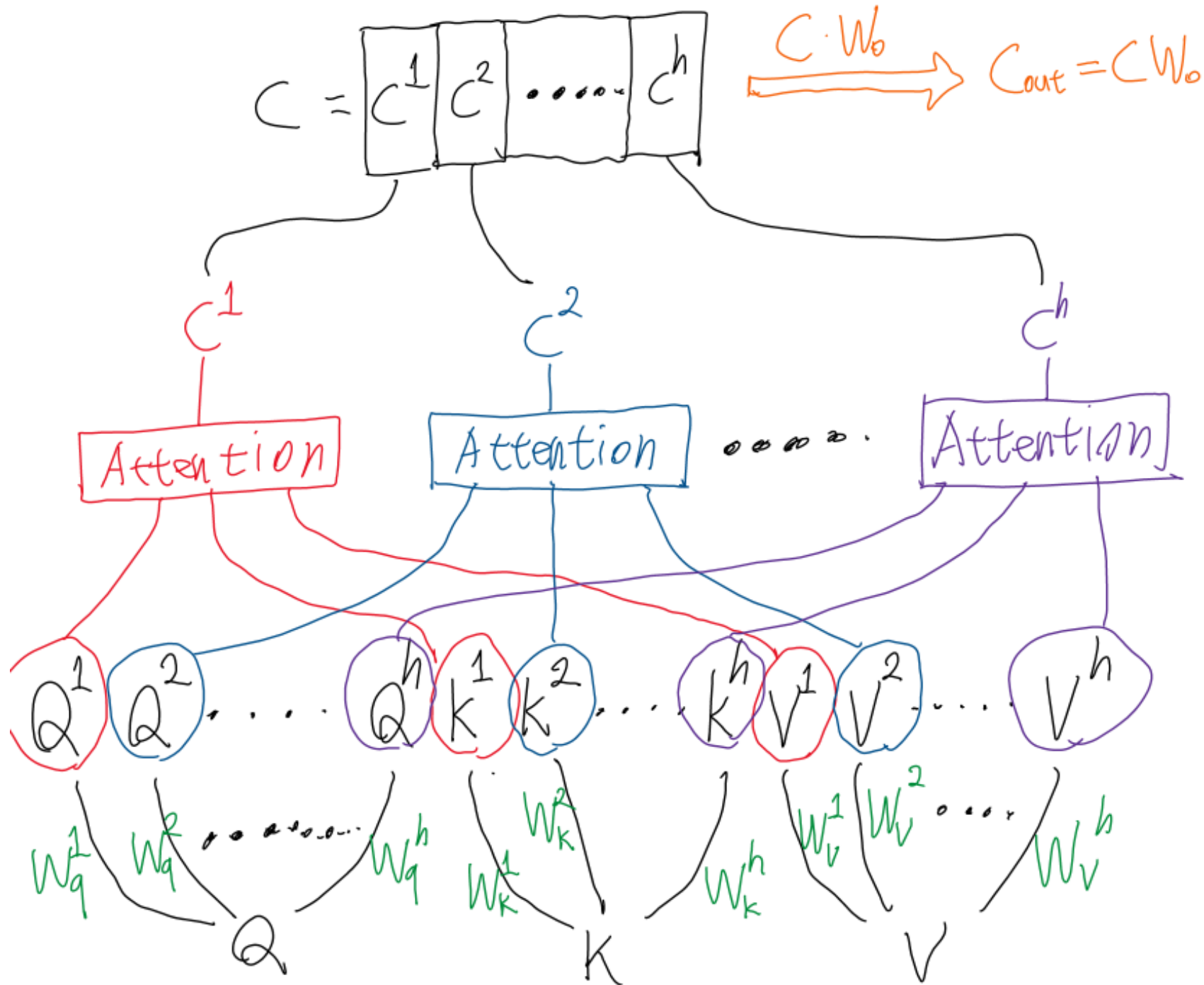
seq2seq

Attention

- Transformer

Q&A Session

공지사항



seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

• Multihead Attention

- 기존의 단일 Attention Head에서는 Q, K, V를 그대로 Scaled Dot Product Attention에 넣

$C = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V$ 로 계산했다.

- Multihead Attention에서는 가중치 행렬 $W_q^{(i)}, W_k^{(i)}, W_v^{(i)}$ 을 각각 h개씩 사용하여 하나의 Attention Head를 h개로 쪼갬다.($i = 1, 2, \dots, h$)

$$Q^{(i)} = QW_q^{(i)} \rightarrow (t, d) * (d, d_k) \rightarrow (t, d_k)$$

$$K^{(i)} = KW_k^{(i)} \rightarrow (T, d) * (d, d_k) \rightarrow (T, d_k)$$

$$V^{(i)} = VW_v^{(i)} \rightarrow (T, d) * (d, d_v) \rightarrow (T, d_v)$$

- $W_q^{(i)}, W_k^{(i)}, W_v^{(i)}$ 의 차원: $(d, d_k), (d, d_k), (d, d_v)$. $d_k = d_v = \left\lfloor \frac{d}{h} \right\rfloor$ 로 정의

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

• Multihead Attention

- i 번째 Attention Head($i = 1, 2, \dots, h$)에 각각 앞에서 구한 $Q^{(i)}, K^{(i)}, V^{(i)}$ 를 넣어서

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

를 계산하여 (t, d_v) 모양의 행렬 h 개를 얻는다. (이 때, $d_k = d_v = \left\lfloor \frac{d}{h} \right\rfloor$)

이를 모두 이어 붙이면 차원이 $(t, h[d/h])$ 인 하나의 행렬 C 를 얻는다.

$$C = \begin{bmatrix} \overset{\rightarrow(1)}{c_1} & \overset{\rightarrow(2)}{c_1} & \dots & \overset{\rightarrow(h)}{c_1} \\ \overset{\rightarrow(1)}{c_2} & \overset{\rightarrow(2)}{c_2} & \dots & \overset{\rightarrow(h)}{c_2} \\ \vdots & \vdots & & \vdots \\ \overset{\rightarrow(1)}{c_t} & \overset{\rightarrow(2)}{c_t} & \dots & \overset{\rightarrow(h)}{c_t} \end{bmatrix}$$

seq2seq

Attention

- Transformer

Q&A Session

공지사항

Transformer

- **Multihead Attention**

- Multihead Attention을 통해 나온 결과는 단일 Attention head를 사용하여 나왔던 결과의 차원 (t, d)와 같아야 한다. 따라서 $W_o \in \mathbb{R}^{h[d/h] \times d}$ 를 곱하여 최종 출력 $C_{out} = CW_o$ 를 얻는다.

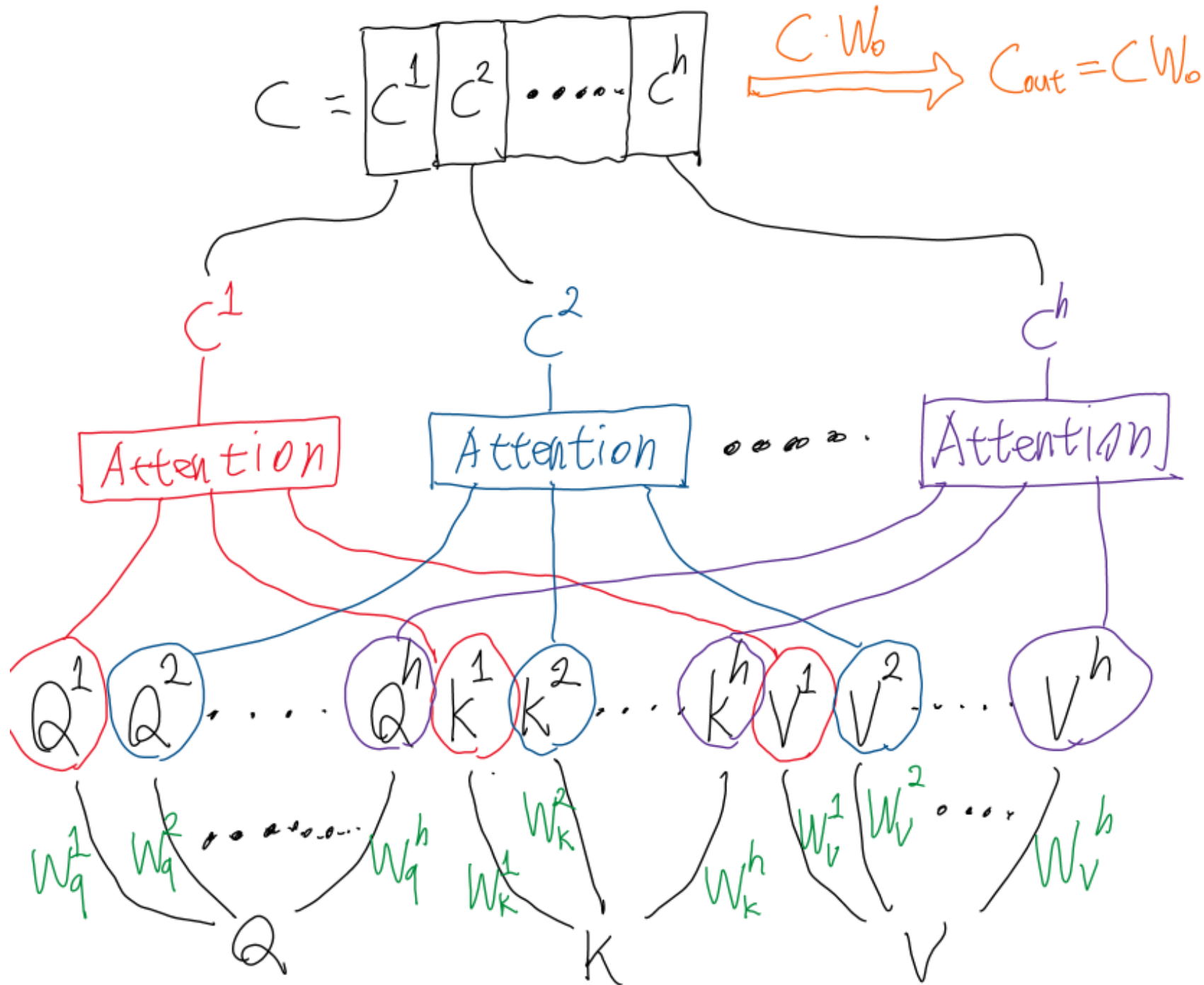
seq2seq

Attention

- Transformer

Q&A Session

공지사항



seq2seq

Attention

- Transformer

Q&A Session

공지사항

Transformer

· Multihead Attention

지금까지의 과정은 하나의 sequence(text 분야에서는 하나의 문장을 의미)를 처리하는 과정을 배운 것이다.

우리는 수많은 sequence를 학습시켜야 한다. 즉, text를 예로 들면 수많은 문장을 학습시켜야 한다.

따라서 하나의 문장을 의미하는 Batch를 한번에 처리할 수 있어야 한다. 즉, N개의 sequence로 이루어진 Batch를 한번에 처리할 것이다.

$$Q: (t, d), K: (T, d), V = (T, d)$$



$$Q: (t, N, d), K: (T, N, d), V = (T, N, d)$$

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

• Multihead Attention

- 기존의 단일 Attention Head에서는 Q, K, V를 그대로 Scaled Dot Product Attention에 넣어

$$C = \text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right)V \text{ 로 계산했다.}$$


- Multihead Attention에서는 가중치 행렬 $W_q^{(i)}, W_k^{(i)}, W_v^{(i)}$ 을 각각 h개씩 사용하여 하나의 Attention Head를 h개로 쪼갬다. (i = 1, 2, ..., h)
- 단일 Attention head와 다른 점은, Q, K, V의 shape에 N 항목이 추가되기 때문에 가중치 행렬의 shape도 변화를 주어야 한다는 것이다.

$$Q^{(i)} = QW_q^{(i)} \rightarrow (t, d) * (d, d_k) \rightarrow (t, d_k)$$

$$K^{(i)} = KW_k^{(i)} \rightarrow (T, d) * (d, d_k) \rightarrow (T, d_k)$$

$$V^{(i)} = VW_v^{(i)} \rightarrow (T, d) * (d, d_v) \rightarrow (T, d_v)$$

$$- d_k = d_v = \left\lfloor \frac{d}{h} \right\rfloor$$


$$\begin{aligned} Q' &= QW_q: (t, N, d) \times (h, d, d_k) \rightarrow (t, N, h, d_k) \\ K' &= KW_k: (T, N, d) \times (h, d, d_k) \rightarrow (T, N, h, d_k) \\ V' &= VW_v: (T, N, d) \times (h, d, d_v) \rightarrow (T, N, h, d_v) \end{aligned}$$

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

· Multihead Attention

- 마찬가지로 dot product로 attention score 계산 후 softmax 적용하여 attention weight를 얻는 과정을 거쳐야 한다. 우선 Q와 K를 dot product하여 E를 구한다.

$$E = Q' \times K': (t, N, h, d_k) \times (T, N, h, d_k) \rightarrow (t, N, h, T)$$

- N과 h는 행렬 연산을 통해서 변하지 않는다. N은 batch, 즉 문장의 개수이고 h는 attention head의 개수이다. 이 둘은 사용자가 지정하는 hyper parameter이다.

(즉, N과 h는 계산을 할 때 무시해서 계산해도 된다.)

- 이렇게 얻은 E를 $\sqrt{d_k}$ 로 나눠주고 softmax를 통과해서 attention weight인 α 계산. 이때 차원은 그대로 (t, N, h, T)

$$\alpha = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)$$

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

• Multihead Attention

-Attention weight와 V를 행렬곱하여 attention head 별 context vector를 얻는다.

$$C' = \alpha \times V': (t, N, h, T) \times (T, N, h, d_v) \rightarrow (t, N, h, d_v)$$

-Attention head 별로 얻어진 context vector를 모두 합쳐서, 단일 attention일 때와 같은 차원의 결과가 나오도록 Linear를 한번 통과시킨다.

$$C' \rightarrow C: (t, N, h, d_v) \rightarrow (t, N, h d_v)$$

$$C_{out} = C \times W_o: (t, N, h d_v) \times (h d_v, d) \rightarrow (t, N, d)$$

- $d_k = d_v = \left\lfloor \frac{d}{h} \right\rfloor$ 임을 잊지 말자.

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

· Masked Multihead Attention

- 트랜스포머의 디코더는 인코더와 다르게 target sequence가 입력되어야 한다.

이것은 상식적으로 말이 안된다. 답을 예측하기 위한 학습을 진행하는데 입력으로 답을 넣어버리면 그냥 답을 외우면 되는 것 아닌가? 그럼에도 target sequence를 입력하는 이유는?

: Teacher Forcing을 위해!

그럼 Teacher Forcing은 무엇이고, 이 것은 어떻게 적용되는 것인가?

seq2seq

Attention

- **Transformer**

Q&A Session

공지사항

Transformer

- **Masked Multihead Attention**

- NLP는 전통적으로 현재 예측값을 가지고 다음 토큰을 예측하는 구조를 가진다. (그러므로 언어에서는 순서가 중요하다.)
- 그런데 이런 구조라면, 첫 번째 토큰의 예측이 잘못되면 이를 가지고 예측하는 다음의 토큰들은 당연히 정답과 거리가 멀어진다.
- 즉, 예측이 올바르지 못하면 그 이후의 토큰 예측은 정상적으로 수행되지 못한다. 이를 방지하는 것이 Teacher Forcing!

seq2seq

Attention

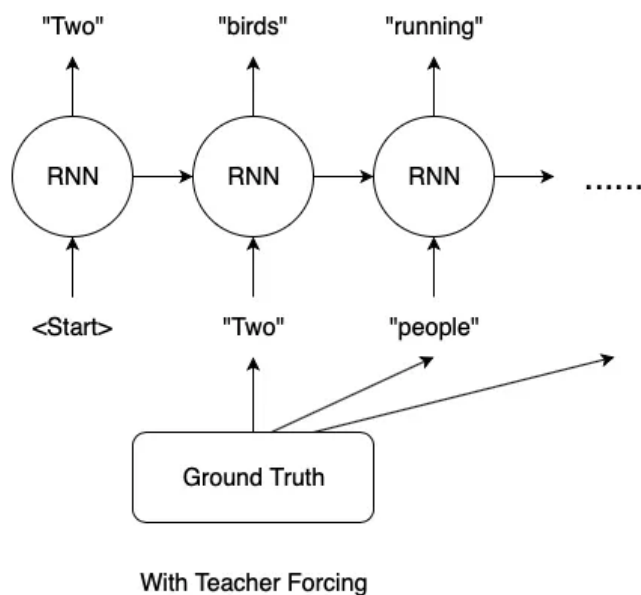
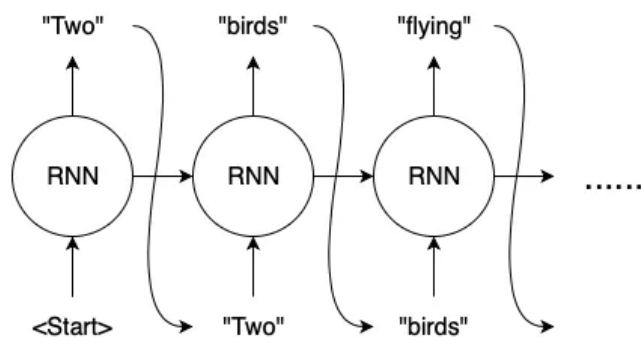
• Transformer

Q&A Session

공지사항

Transformer

· Teacher Forcing



- 잘못된 예측이 있을 경우, 다음 예측에 사용될 입력 값으로 올바른 예측 값을 넣는 것이 Teacher Forcing

- 트랜스포머의 디코더에서도 Training 과정에서 Teacher Forcing 기법을 활용하기 위해 target sequence를 입력해준다.

- 그런데, 이 과정에서 target sequence의 모든 원소를 그 값 그대로 입력해주면 신경망은 그 값을 그대로 외우게 될 것이다. 이를 방지하기 위해 입력되는 target sequence에 제한을 걸어준다.

- t번째 토큰을 예측하고자 한다면, t+1번째 이후 토큰들에 대한 정보는 알면 안된다. 이를 위해 mask를 attention 층을 거친 target sequence에 씌워 준다.

seq2seq

Attention

• Transformer

Q&A Session

공지사항

Transformer

· Masked Multihead Attention

- 예를 들어, target sequence에 self-attention을 적용한 결과가 다음과 같다고 하자.

```
np.array([[0.1, 0.7, 0.2],  
          [0.4, 0.2, 0.4],  
          [0.1, 0.8, 0.1]])
```

- 위 배열의 첫 번째 행을 보면, 첫 토큰을 예측할 때 두 번째 토큰의 attention 값이 0.7로 상당히 많은 부분 관여한다고 할 수 있다. 이러한 영향은 바람직하지 않다.
- 첫 토큰의 경우 첫 토큰만이, 두 번째 토큰의 경우 첫 토큰과 두 번째 토큰의 영향만을 받아야 함(즉, 토큰 예측은 현재 토큰과 그 이전 토큰들만을 가지고 예측해야 한다.)
- 따라서, 다음과 같이 0으로 마스킹 해주어 현재 토큰 이후에 나오는 토큰들의 영향을 제거해야 한다.

```
np.array([[0.1, 0, 0],  
          [0.4, 0.2, 0],  
          [0.1, 0.8, 0.1]])
```

seq2seq

Attention

- Transformer

Q&A Session

공지사항

Transformer

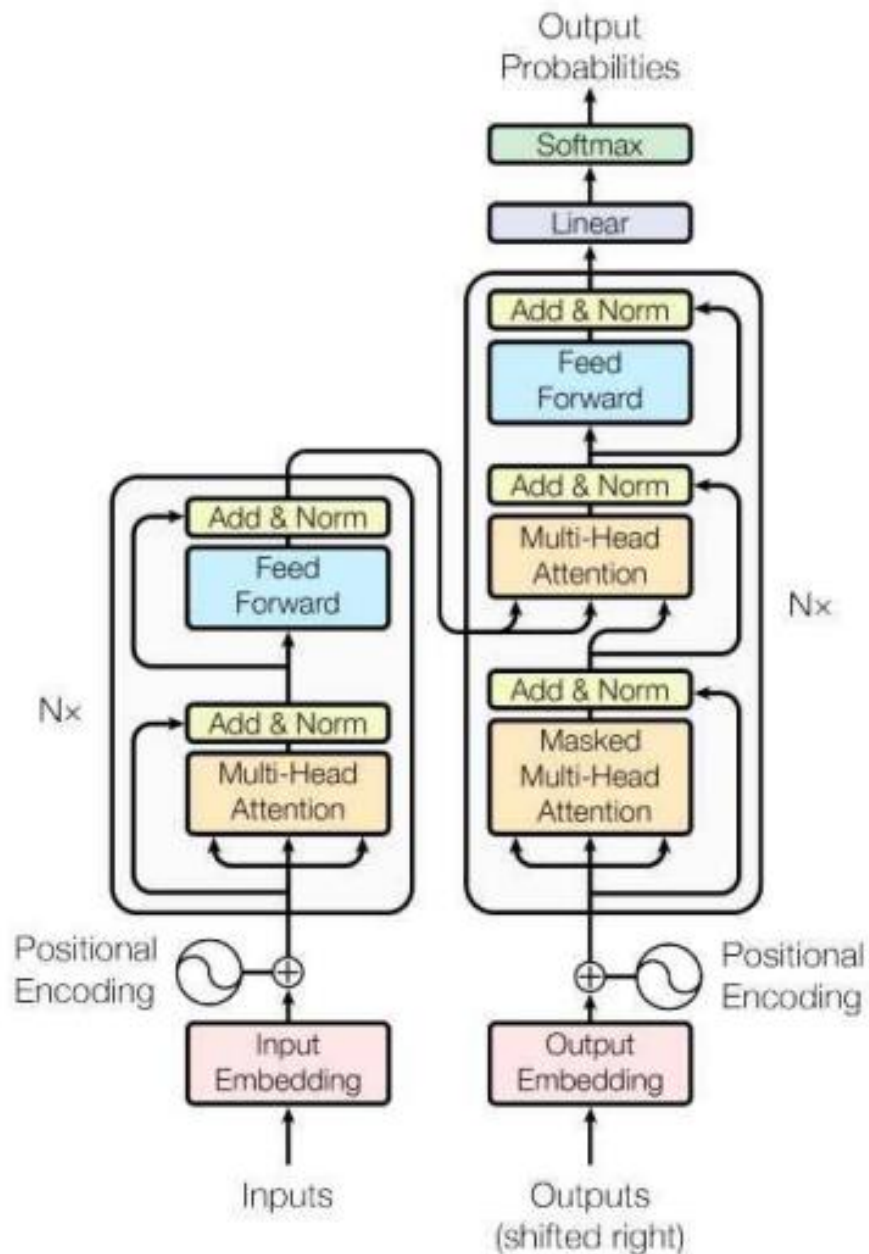


Figure 1: The Transformer - model architecture.



감사합니다

