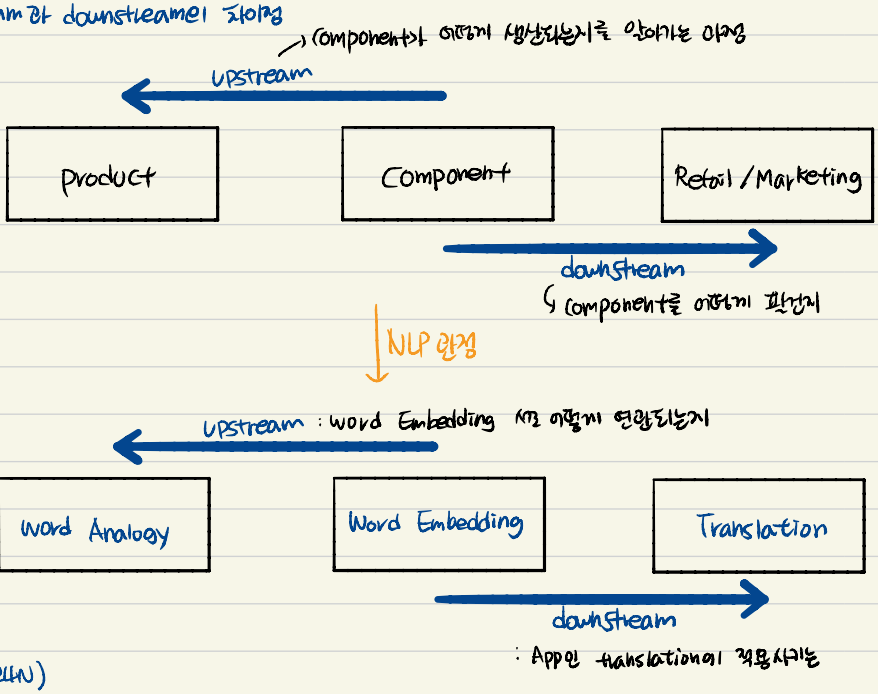
김동화 - Transformer & Bert 유튜브 영상보고 transformer 추가공부

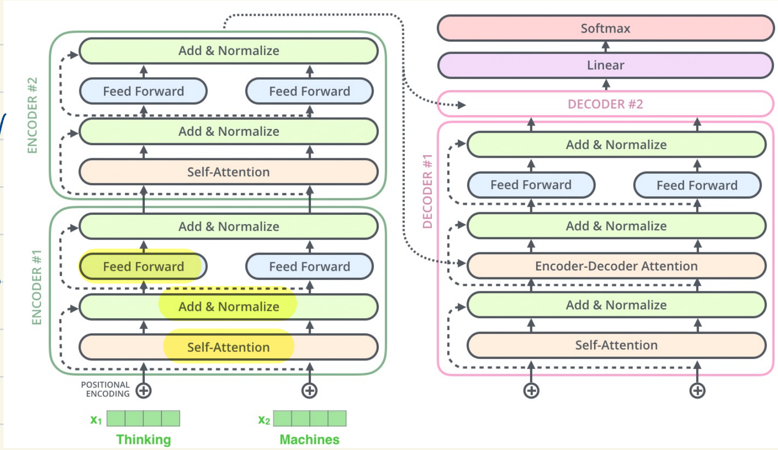
- Transformer, bert → downstream task 목적
- downstream 이란?
  - upstream과 downstream의 차이점



(In CS224N)

- upstream : Intrinsic (내적)한 관점
- downstream : Extrinsic (외적)한 관점

· Transformer



n개의 encoder를 사용.

· 모델의 목적 ┌ Downstream Task for NLP
└ Translation

· 데이터의 구성 – Pair Sentence translation은 꼭 pair sentence!

· Attention 방법 ┌ key, Query, Value
└ K, Q, V는 Encoder, Decoder에서 각각 생성됨

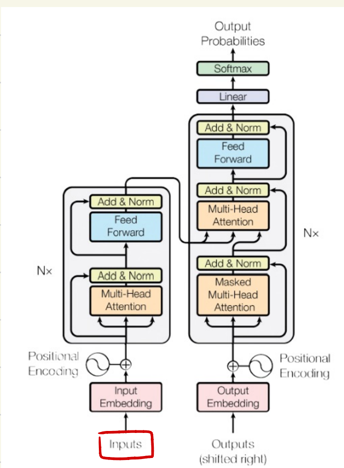


Figure 1: The Transformer - model architecture.

· input을 어떻게 넣을까. : Tocken Embedding

– Max_length = 6 일때 → max sequence length가 6일때.

문장의 길이가 다 다를수 있다.
하지만 고정된 길이의 tensor가 학습이 됨
→ 일정값을 정해놓고 그거보다 크면 자르고
작으면 padding 이라는 token을 추가해서

┌ Sent1 : i am looking for happiness
├ Sent2 : i am suhyon
└ Sent3 : i am @#$

→ 어떻게 vocabulary를 정의할 수 있는가.

– 인코더, 디코더를 각각 vocab를 정의

– 4개의 token을 추가 : <PAD>, <UNK>, <EOS>, <BOS>

<PAD>: 길이를 맞추기위한
<UNK> (2.): 처음보는 것에 대한 임의의 token
<EOS>: 문장의 끝을 알리는
<BOS>: 디코더에서 사용되는

- Encoder, max_length = 6일때,

  문장의 끝 token

  Sent1: I am looking for happiness {x_1, x_2, x_3, x_4, x_5, </s>}

  Sent2: I am SUhyun {x_1, x_2, x_3, </s>, <PAD>, <PAD>}

  Sent3: I am @#$ {x_1, x_2, <UNK>, </s>, <PAD>, <PAD>}

  → 모두 같은 차원으로 만듦

- Decoder, max_length = 6일때,

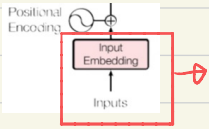
  문장의 token들 / 끝이 인식되면 학습종료 0

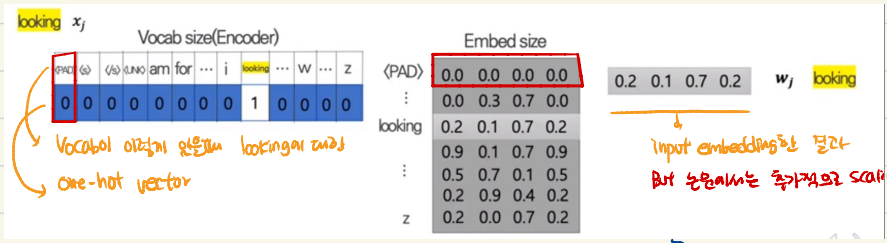
  Sent1: 나는 행복을 찾았다 {<s>, x_1, x_2, x_3, x_4, </s>}

  Sent2: 나는 수현이다 {<s>, x_1, x_2, </s>, <PAD>, <PAD>}

  Sent3: 나는 @#$ {<s>, x_1, <UNK>, </s>, <PAD>, <PAD>}

→ I am looking for happiness {x_1, x_2, x_3, x_4, x_5, </s>}





□ : <PAD> 값의 경우는 근본적에서 학습이라지 않게 의미없는 값으로 간주시키기!

- Token embedding scaling

$0.2\ 0.1\ 0.7\ 0.2 \times \frac{1}{\sqrt{embed\_size}}$: 좀 더 gradient를 잘.. 하기 위함

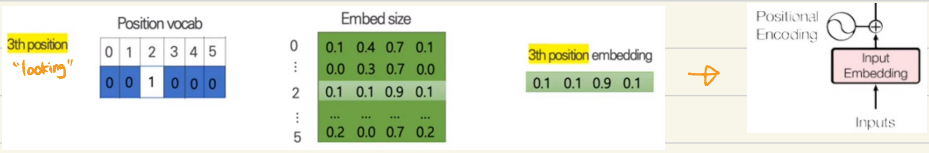
- positional encoding

: 입력값의 상대적 위치를 표현하기 위해 사용

방법 2가지 ① Xavier initialize

② Sinusoid initialize

① Xavier initialize



: positional 값을 one-hot vector를 weight matrix로 곱해서 position embedding 생성

: 생성된 position embedding을 input Embedding의 output과 더해줌

② Sinusoid initializer

$$PE_{(pos,2i)} = \sin(pos / 10^{4*2i/d_{model}})$$

$$PE_{(pos,2i+1)} = \cos(pos / 10^{4*2i/d_{model}})$$

d_{model} : embedding size

i : embedding feature의 인덱스

pos : 단어의 인덱스 → 총 6개의 index

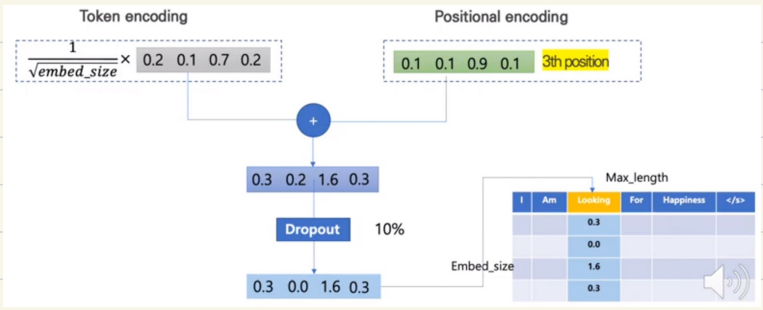
```
import numpy as np
import matplotlib.pyplot as plt

max_length = 512
embed_size = 1024
position_enc = np.array([
    [pos / np.power(10000, 2*i/embed_size) for i in range(embed_size)]
    if pos != 0 else np.zeros(embed_size) for pos in range(max_length)])

position_enc[1:, 0::2] = np.sin(position_enc[1:, 0::2]) # dim 2i
position_enc[1:, 1::2] = np.cos(position_enc[1:, 1::2]) # dim 2i+1

img = plt.imshow(position_enc, cmap='hot', interpolation='nearest', aspect='auto')
cmap = plt.get_cmap('hot')
plt.colorbar(img, cmap=cmap)
plt.show()
```

- Dropout

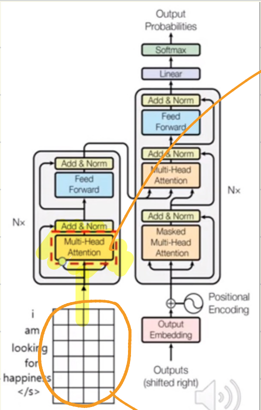


: feature의 개수를 줄이는 것. → overfitting을 방지하기 위해

↳ layer가 많아질때 발생하는 대표적인 문제점.

각 단어의 대표성도

- Multi-head Attention



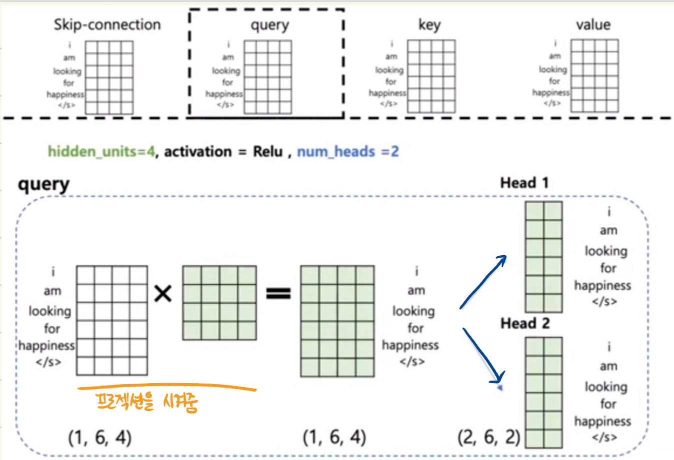
→ 행렬이 input으로 들어간다.

: 하나가 들어가는 것처럼 보이지만 4개로 복사되어 들어감

: 각각은 Skip-Connection / query / key / value

→ input embedding + position embedding 으로 계산된 결과 tensor

① Query



프로젝션을 시켜줌

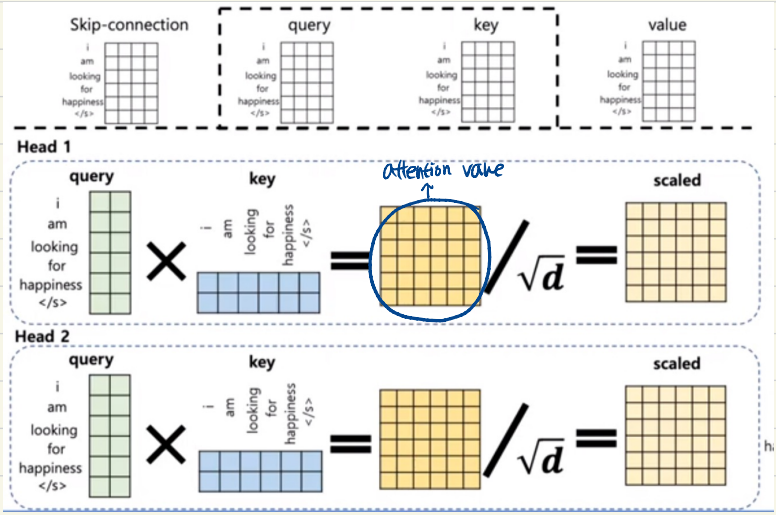
쪼개는 이유 ≒ 한번에 해서 반으로 나눈 이유

: attention weight를 좀 더 다양하게 가져서 다양한 시각으로 보기위해

② key
③ Value

} Query와 같은 방식으로!

Head가 2개 → 각각의 Q, K, V

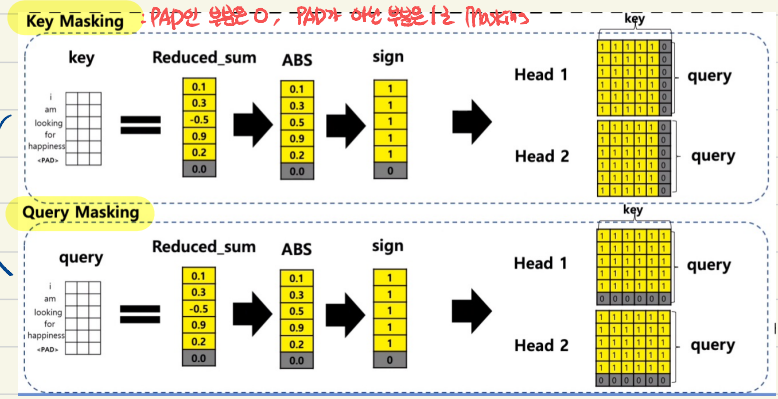


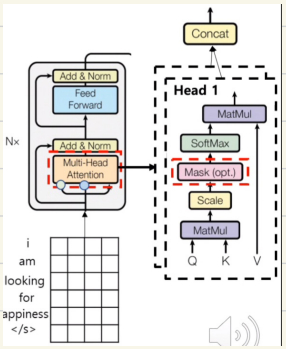
: Query와 key^T를 연산을 통해 Attention value/weight를 구함

But, hidden size가 커질수록 값은 ↑ 값이 커지면 gradient가 vanishing 대략 loss 발생 (기울기값이 사라지는 문제)

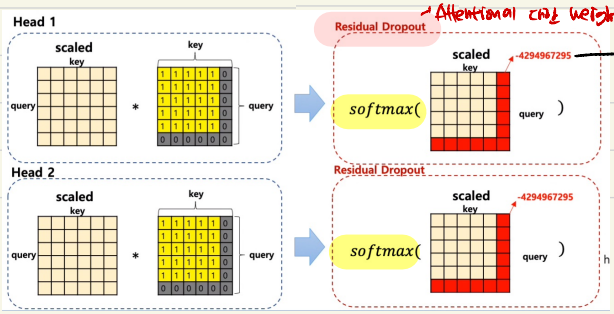
그래서, $/\sqrt{d}$ 를 해줌 ↳ scaling 해주는

*만약 Sequence에 <PAD>가 있는 경우.



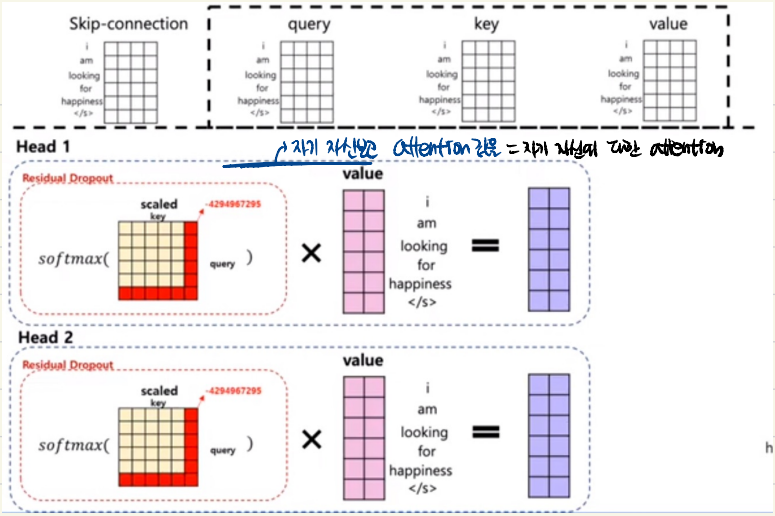


이 둘을 융합하게 되면 <PAD>가 아닌 부분에만 1인 key, Query Masking을 완료 후 0 → 그걸 Scaled attention weight에 적용

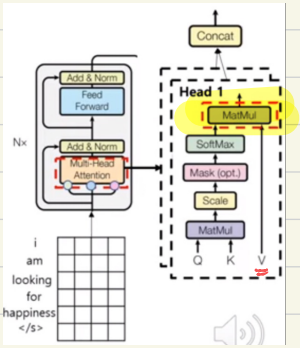


Attention에 대한 weight를 다양하게 해석하려

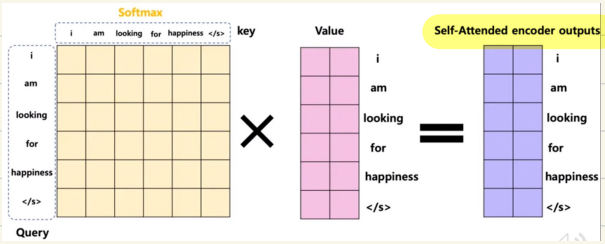
→ 이 부분이 0 이지만 굉장히 큰 음수값을 넣어줌 Softmax를 취할때 → 이 부분의 값이 0과 가깝게 나오게 하기위해



↑자기 자신한테 Attention 값은 = 자기 제외한 다른 Attention



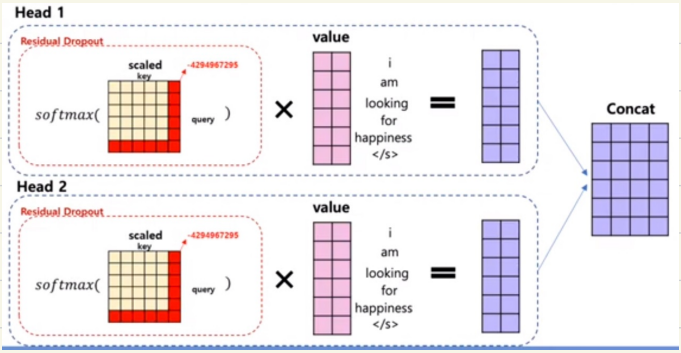
↓더 계산해

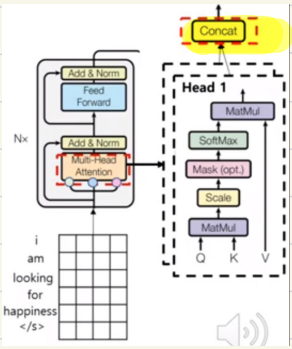


→자기 자신에게 어떠나 더 중요한지 가중치가 달라짐

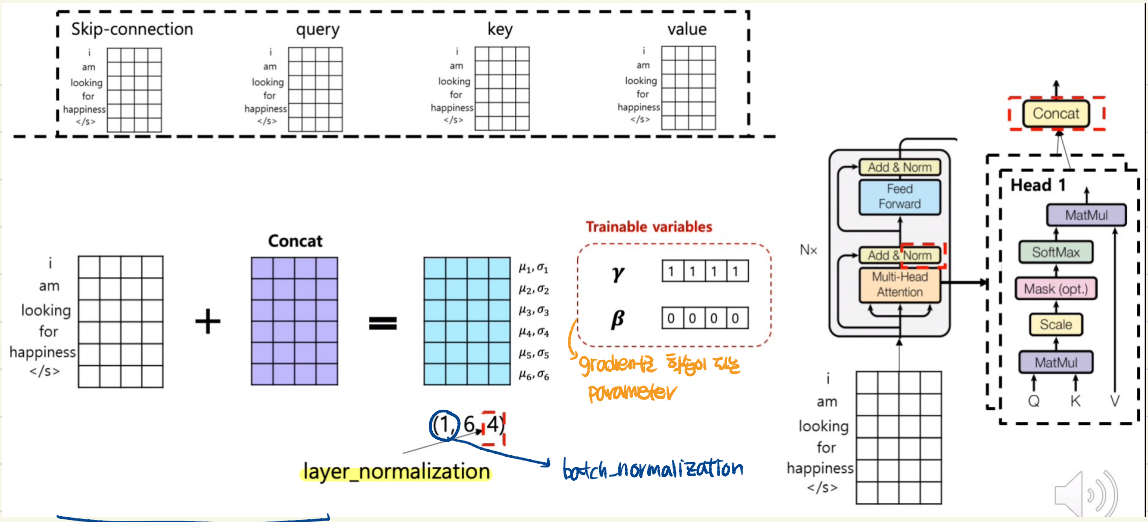
↓

Self-Attention한 output들을 Concat!





• Add & Norm



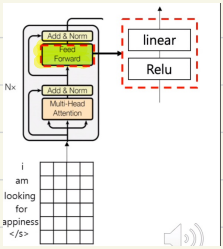
Concat한 행렬을 원래의 input과 더함 : Add

• Feed Forward

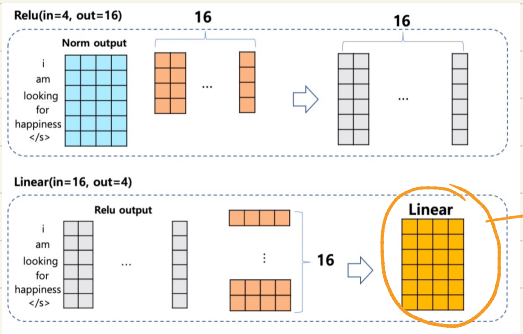
: Add & Norm한 결과에 대해서 해주는

방식 2가지 ① Relu function과 2개 linear

② convolution

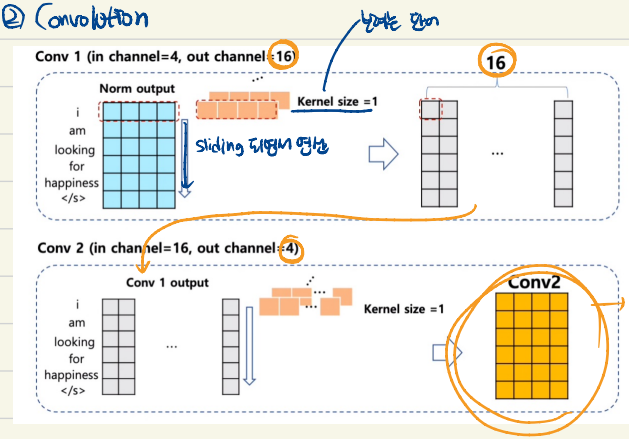


① Relu Function → linear



→ 여기로 Norm output과 더해지게 됨

② Convolution



· Add & Norm