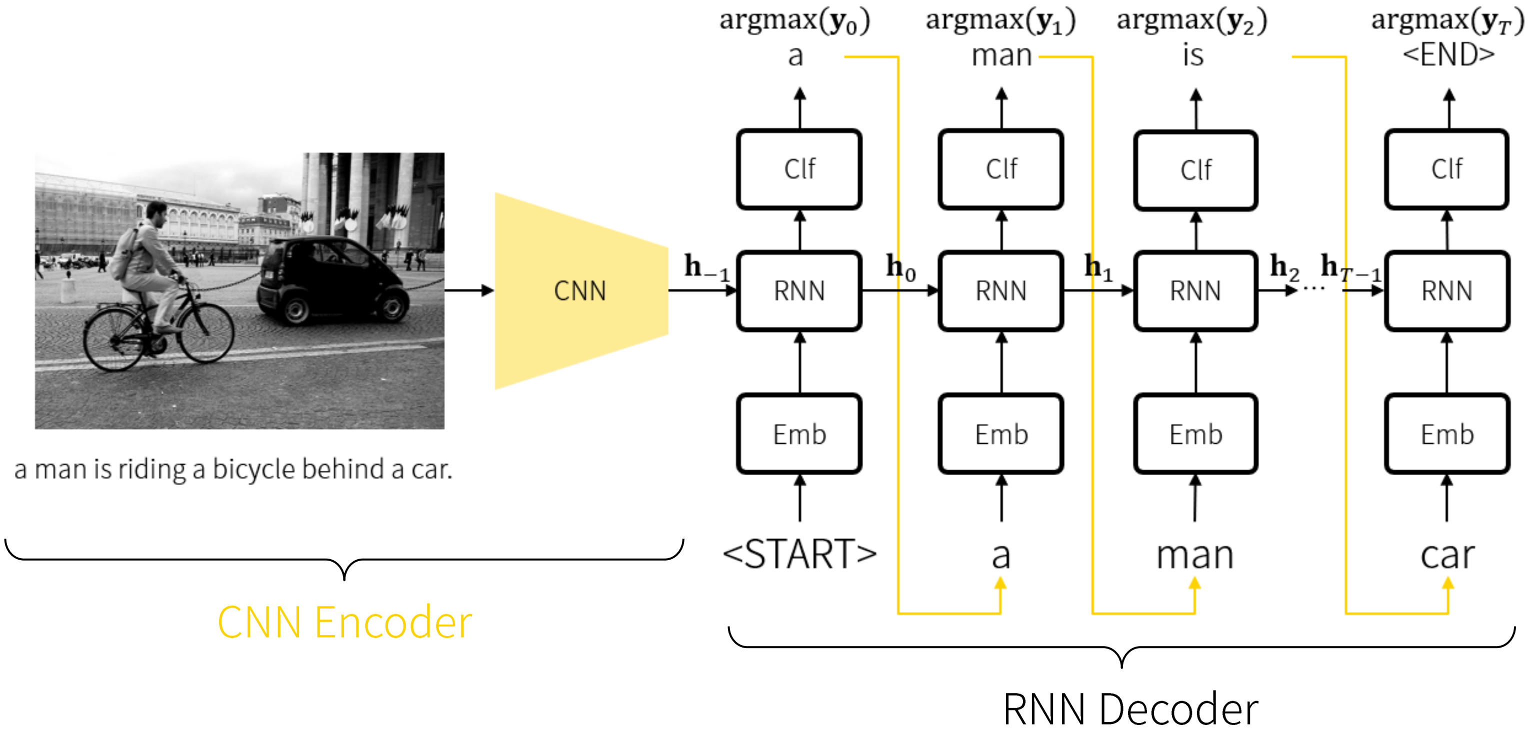
Transformer

The Parts of Transformer and HuggingFace

조준우
metamath@gmail.com

Sequence to Sequence Models

- Encoder-Decoder model

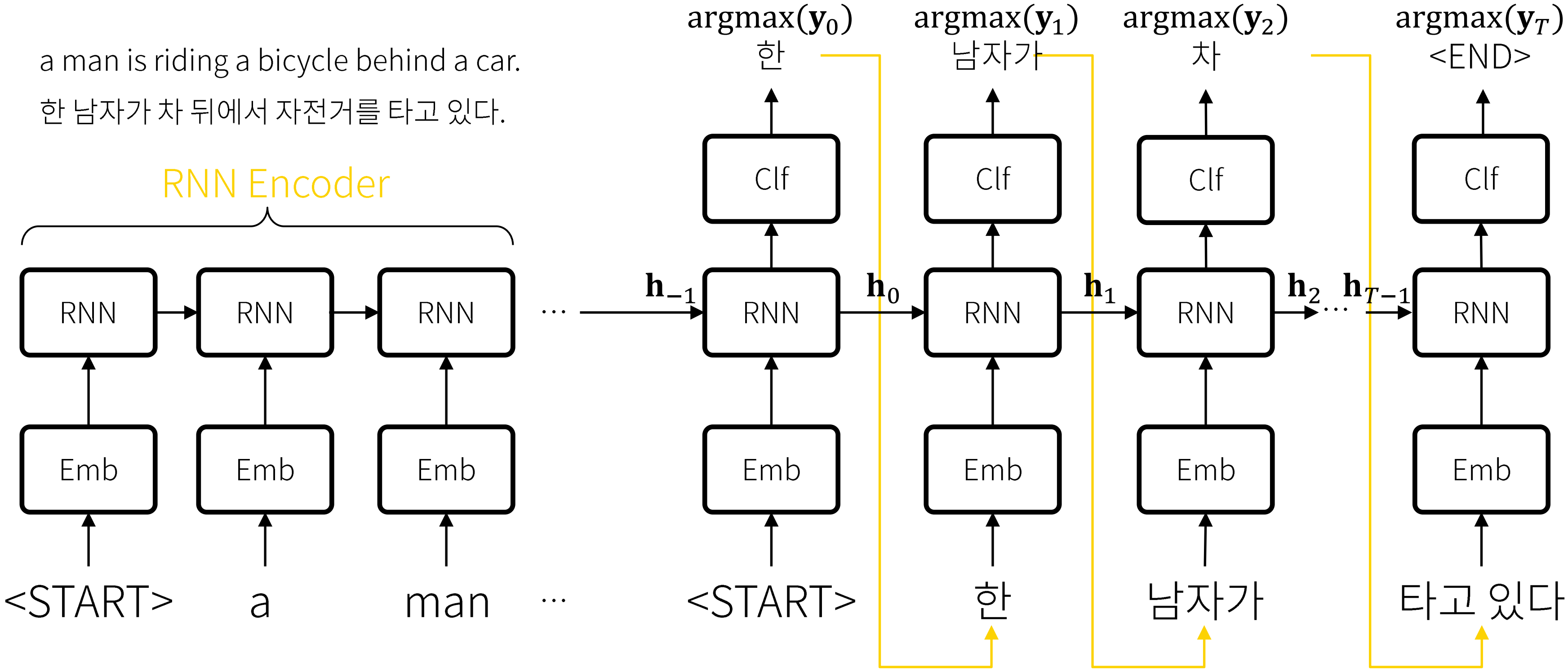


Sequence to Sequence Models

- Encoder-Decoder model

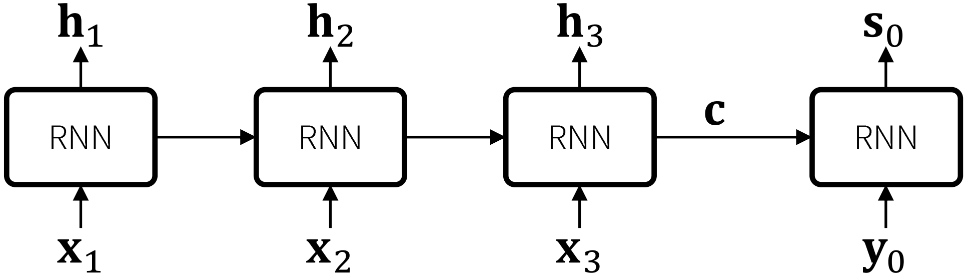
a man is riding a bicycle behind a car.

한 남자가 차 뒤에서 자전거를 타고 있다.

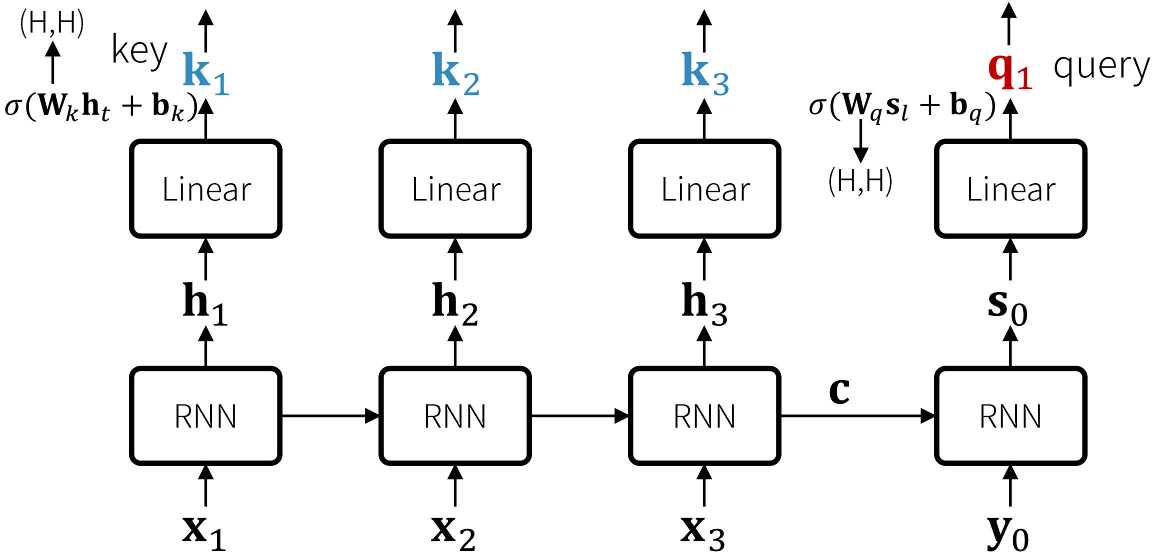


Attention

첫 번째 출력을 잘 만들기 위해 $\mathbf{h}_1, \mathbf{h}_2, \mathbf{h}_3$ 중에 어떤 것을 더 참조하면 좋을까?

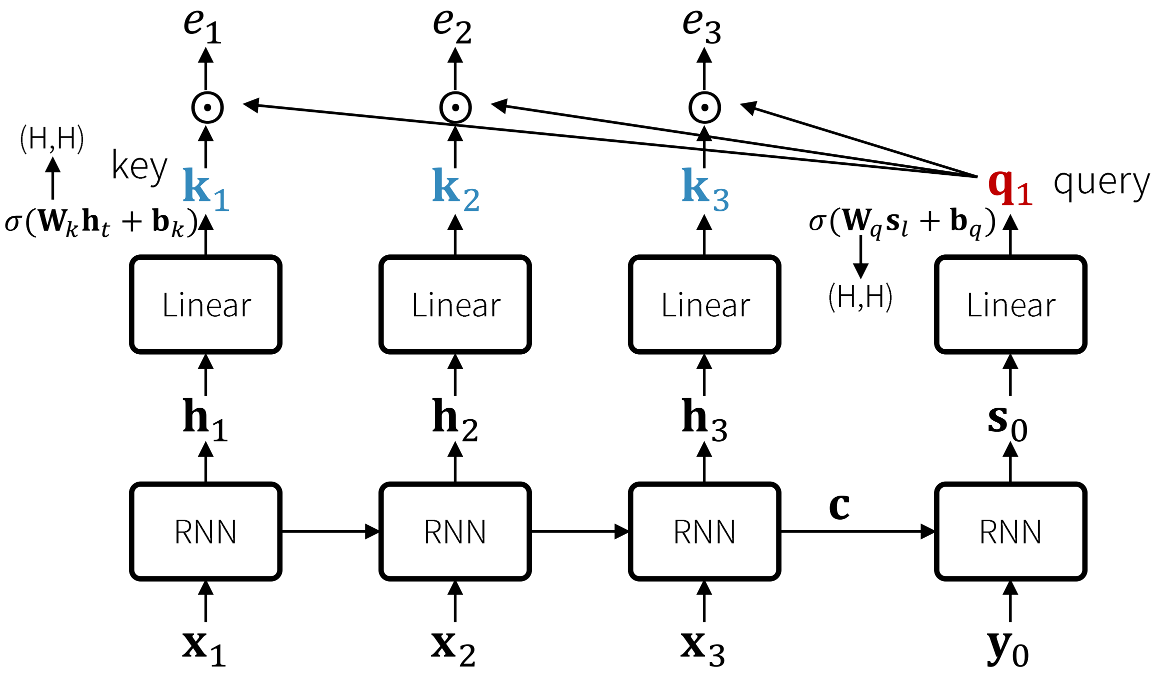


Attention



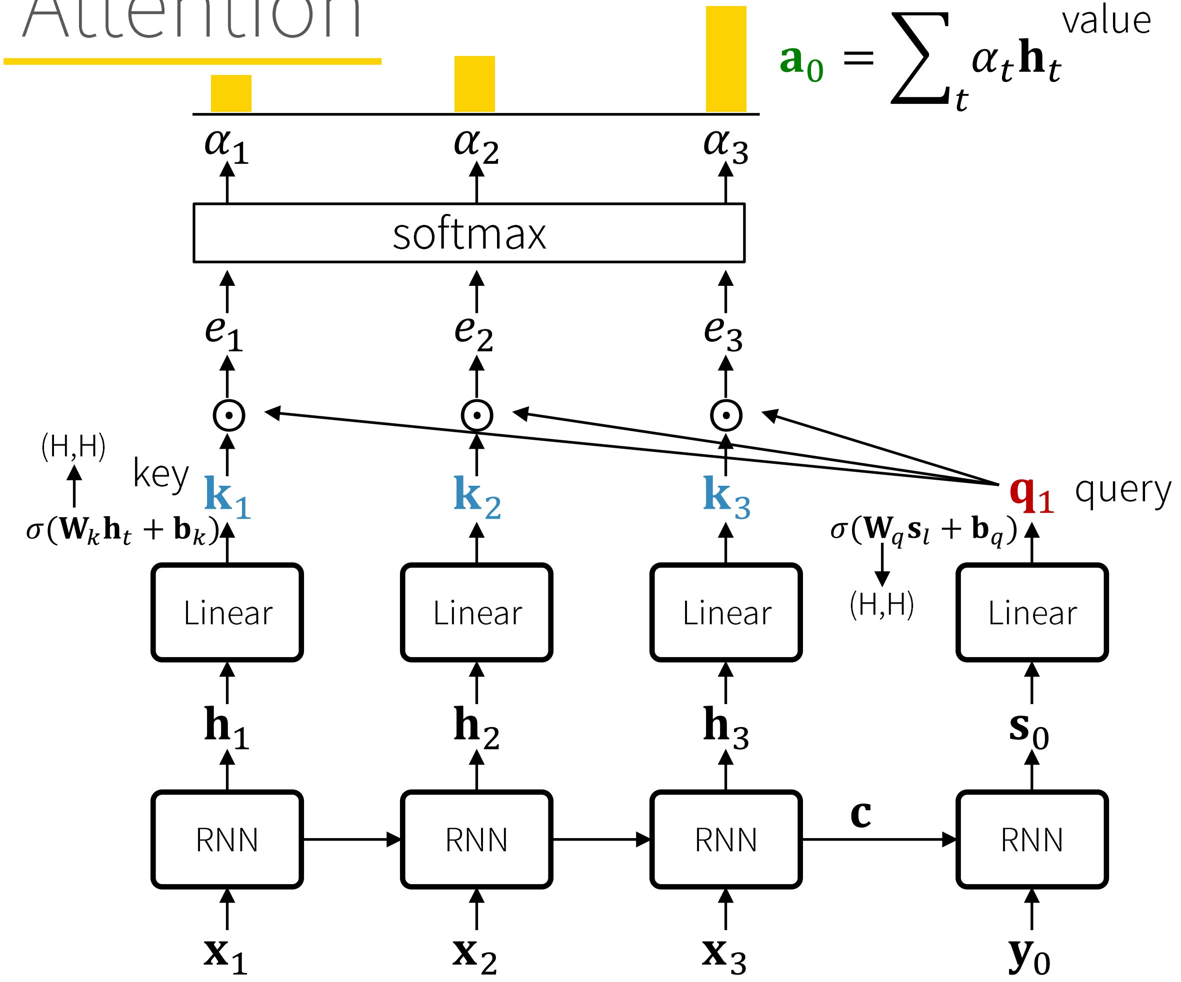
(H,H)
key
$\mathbf{k}_1$
$\sigma(\mathbf{W}_k \mathbf{h}_t + \mathbf{b}_k)$
$\mathbf{k}_2$
$\mathbf{k}_3$
$\mathbf{q}_1$
query
$\sigma(\mathbf{W}_q \mathbf{s}_l + \mathbf{b}_q)$
(H,H)
Linear
Linear
Linear
Linear
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{s}_0$
RNN
RNN
RNN
$\mathbf{c}$
RNN
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{y}_0$

Attention



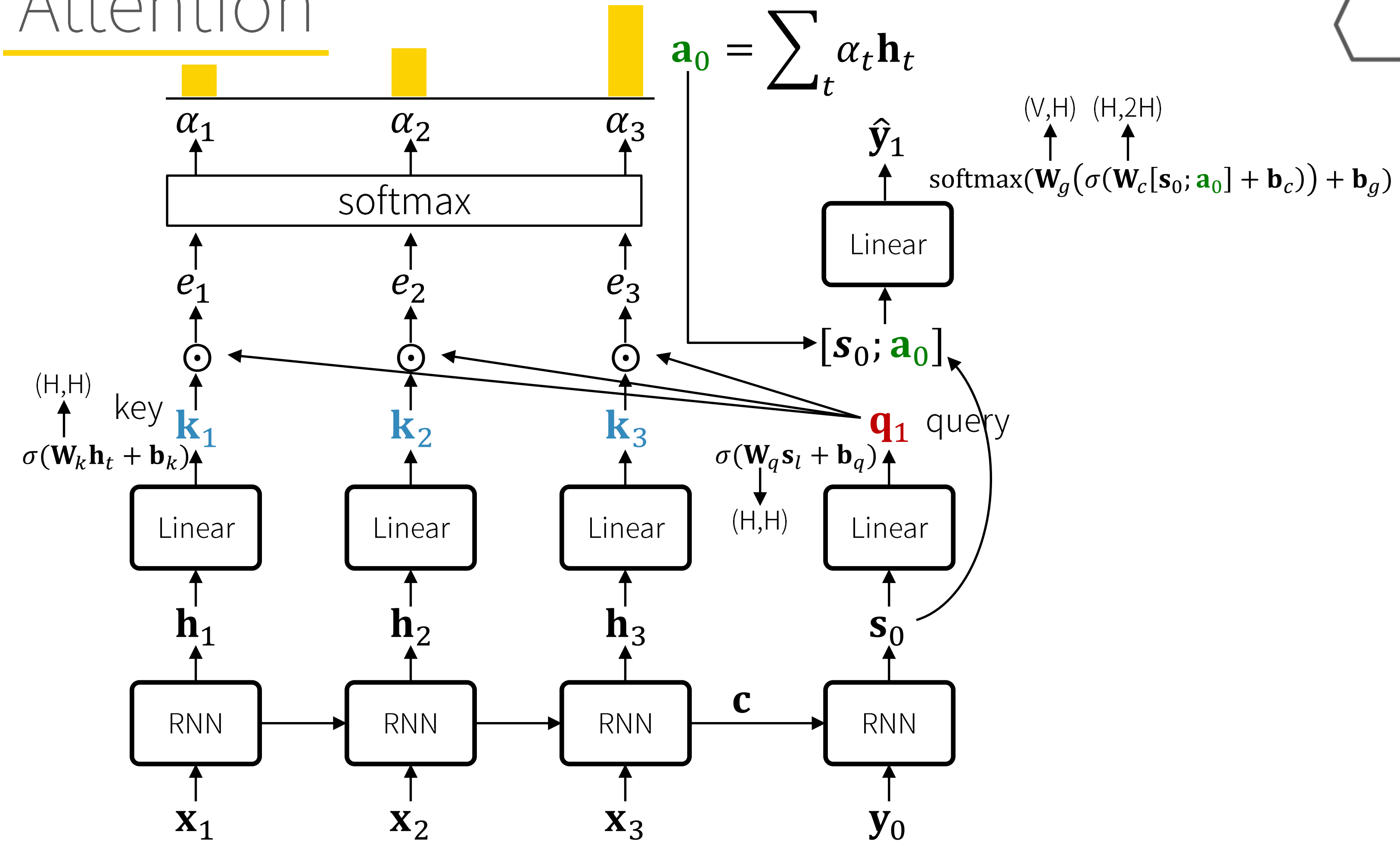
e_1
e_2
e_3
(H,H)
key
$\mathbf{k}_1$
$\mathbf{k}_2$
$\mathbf{k}_3$
$\mathbf{q}_1$
query
$\sigma(\mathbf{W}_k \mathbf{h}_t + \mathbf{b}_k)$
$\sigma(\mathbf{W}_q \mathbf{s}_l + \mathbf{b}_q)$
(H,H)
Linear
Linear
Linear
Linear
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{s}_0$
RNN
RNN
RNN
$\mathbf{c}$
RNN
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{y}_0$

Attention



$\mathbf{a}_0 = \sum_t \alpha_t \mathbf{h}_t$
value
α_1
α_2
α_3
softmax
e_1
e_2
e_3
(H,H)
key
$\mathbf{k}_1$
$\mathbf{k}_2$
$\mathbf{k}_3$
$\mathbf{q}_1$
query
$\sigma(\mathbf{W}_k\mathbf{h}_t + \mathbf{b}_k)$
$\sigma(\mathbf{W}_q\mathbf{s}_l + \mathbf{b}_q)$
(H,H)
Linear
Linear
Linear
Linear
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
$\mathbf{s}_0$
RNN
RNN
RNN
$\mathbf{c}$
RNN
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{y}_0$

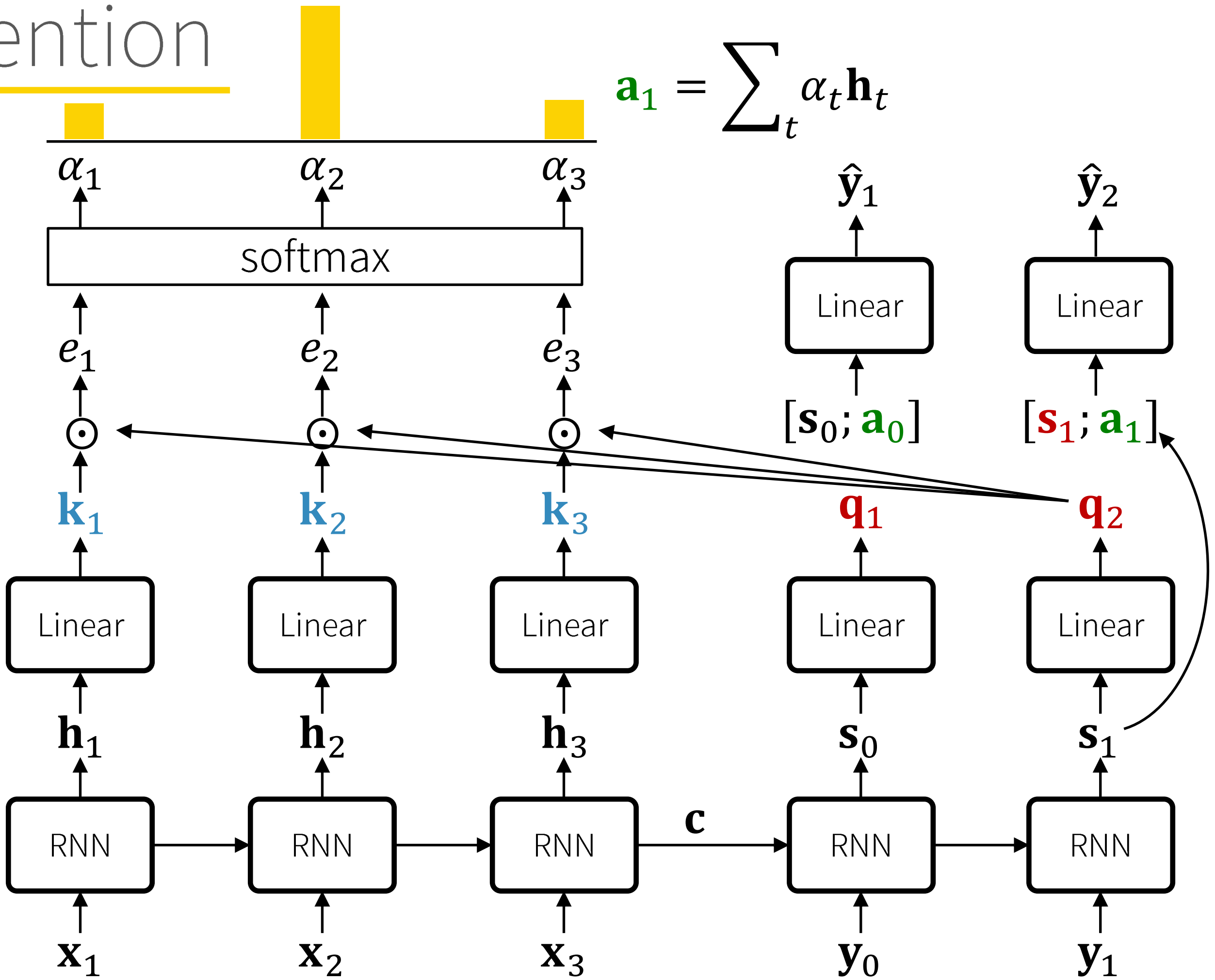
Attention



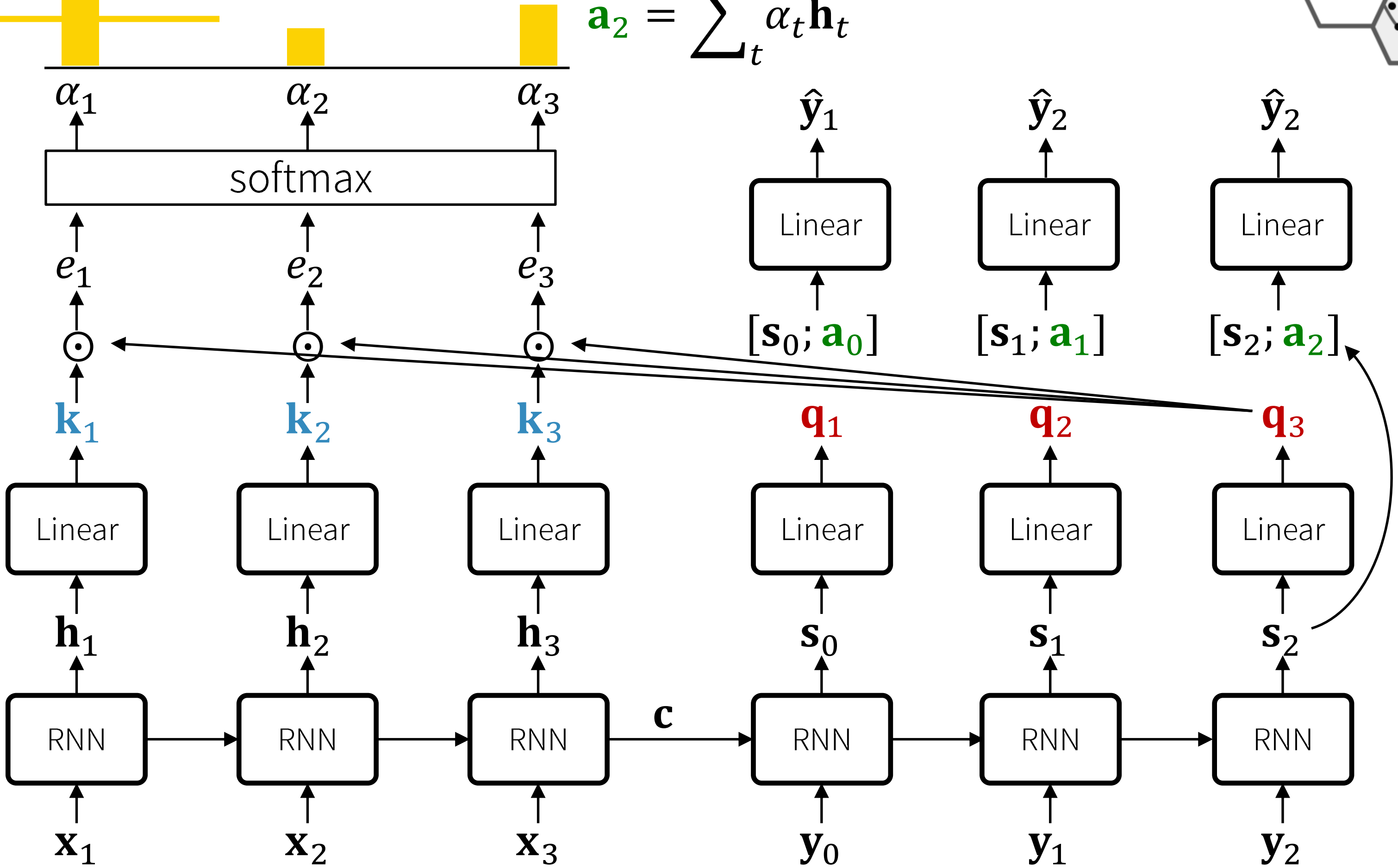
α_1 α_2 α_3
softmax
e_1 e_2 e_3
$\odot$ $\odot$ $\odot$
(H,H)
key
$\mathbf{k}_1$ $\mathbf{k}_2$ $\mathbf{k}_3$
$\sigma(\mathbf{W}_k \mathbf{h}_t + \mathbf{b}_k)$
Linear
Linear
Linear
$\mathbf{h}_1$ $\mathbf{h}_2$ $\mathbf{h}_3$
RNN
RNN
RNN
$\mathbf{x}_1$ $\mathbf{x}_2$ $\mathbf{x}_3$
$\mathbf{c}$
$\mathbf{a}_0 = \sum_t \alpha_t \mathbf{h}_t$
(V,H) (H,2H)
$\hat{\mathbf{y}}_1$
$\mathrm{softmax}(\mathbf{W}_g(\sigma(\mathbf{W}_c[\mathbf{s}_0; \mathbf{a}_0] + \mathbf{b}_c)) + \mathbf{b}_g)$
Linear
$[\mathbf{s}_0; \mathbf{a}_0]$
$\mathbf{q}_1$ query
$\sigma(\mathbf{W}_q \mathbf{s}_l + \mathbf{b}_q)$
(H,H)
Linear
$\mathbf{s}_0$
RNN
$\mathbf{y}_0$

Attention

$$\mathbf{a}_1 = \sum_t \alpha_t \mathbf{h}_t$$



Attention



$\mathbf{a}_2 = \sum_t \alpha_t \mathbf{h}_t$
α_1
α_2
α_3
softmax
e_1
e_2
e_3
$\mathbf{k}_1$
$\mathbf{k}_2$
$\mathbf{k}_3$
Linear
Linear
Linear
$\mathbf{h}_1$
$\mathbf{h}_2$
$\mathbf{h}_3$
RNN
RNN
RNN
$\mathbf{x}_1$
$\mathbf{x}_2$
$\mathbf{x}_3$
$\mathbf{c}$
$\hat{\mathbf{y}}_1$
$\hat{\mathbf{y}}_2$
$\hat{\mathbf{y}}_2$
Linear
Linear
Linear
$[\mathbf{s}_0; \mathbf{a}_0]$
$[\mathbf{s}_1; \mathbf{a}_1]$
$[\mathbf{s}_2; \mathbf{a}_2]$
$\mathbf{q}_1$
$\mathbf{q}_2$
$\mathbf{q}_3$
Linear
Linear
Linear
$\mathbf{s}_0$
$\mathbf{s}_1$
$\mathbf{s}_2$
RNN
RNN
RNN
$\mathbf{y}_0$
$\mathbf{y}_1$
$\mathbf{y}_2$

Transformer

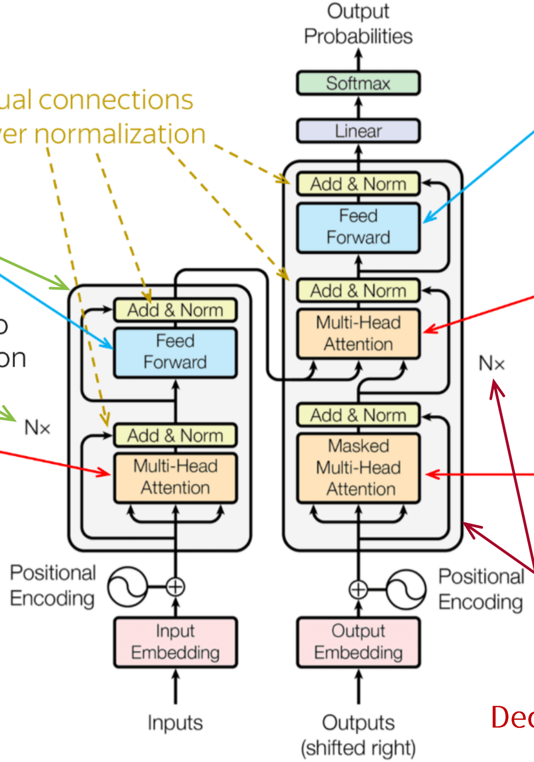
Encoder=(N * Encoder layer)

Residual connections and layer normalization

Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Encoder self-attention: tokens look at each other

queries, keys, values are computed from encoder states



Feed-forward network: after taking information from other tokens, take a moment to think and process this information

Decoder-encoder attention: target token looks at the source

queries – from decoder states; keys and values from encoder states

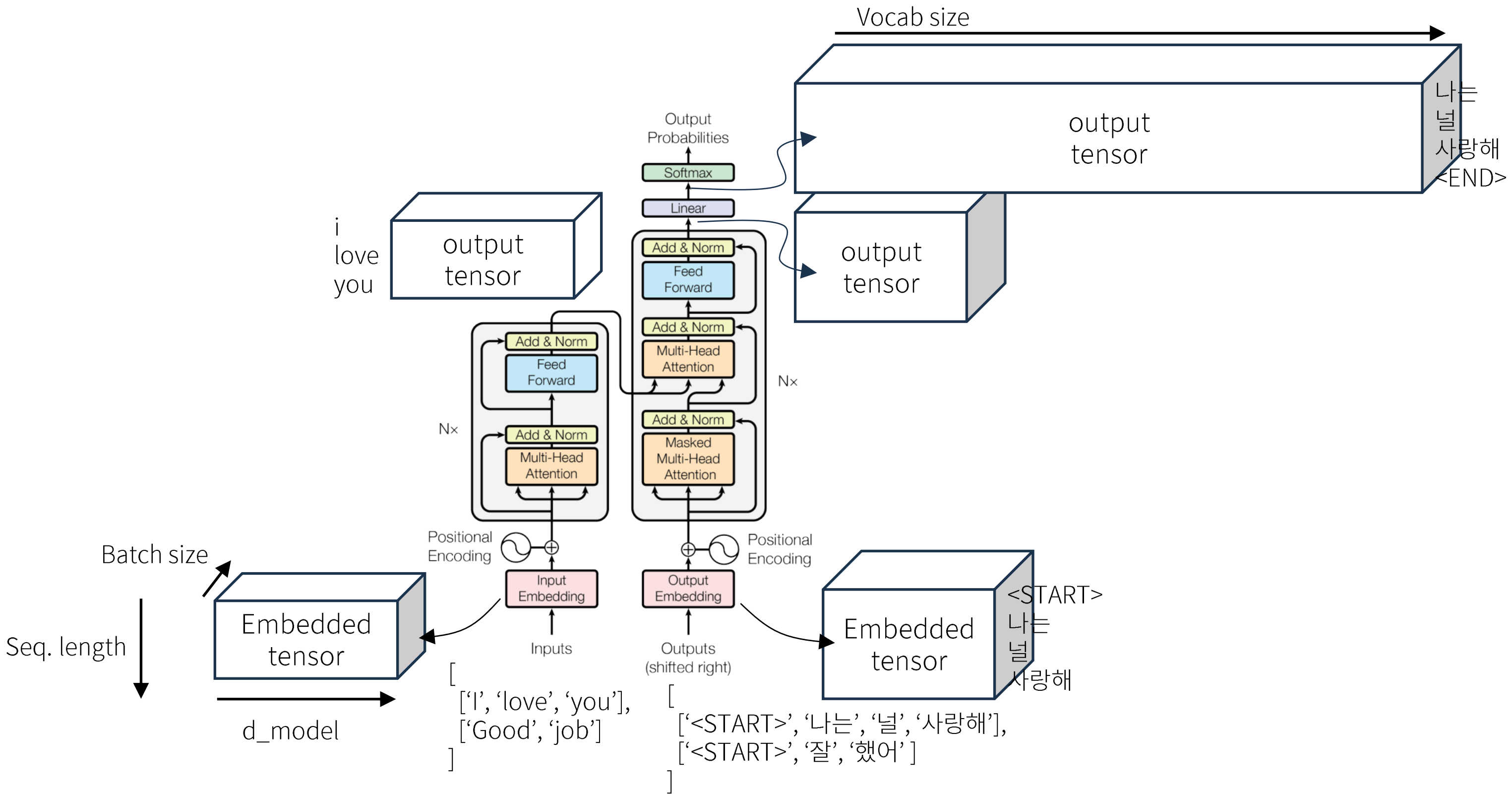
Decoder self-attention (masked): tokens look at the previous tokens

queries, keys, values are computed from decoder states

Decoder=(N * Decoder layer)

https://www.rksmusings.com/2021/07/13/getting-started-with-google-bert-book-review/

Overview

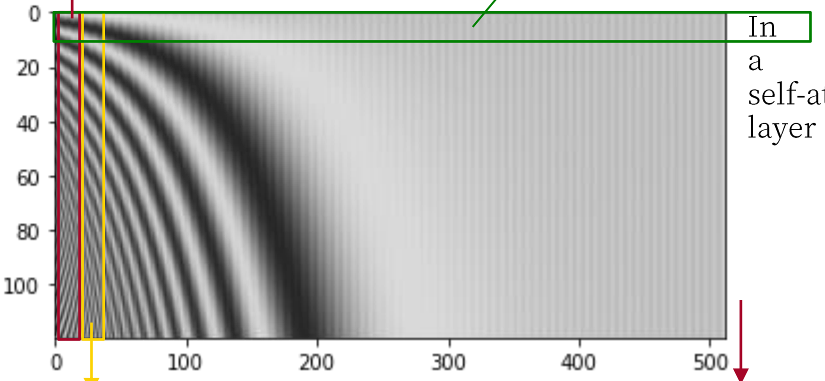

Vocab size
output
tensor
나는
널
사랑해
<END>
Output
Probabilities
Softmax
Linear
i
love
you
output
tensor
output
tensor
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Add & Norm
Masked
Multi-Head
Attention
Nx
Add & Norm
Feed
Forward
Add & Norm
Multi-Head
Attention
Nx
Positional
Encoding
Positional
Encoding
Input
Embedding
Output
Embedding
Inputs
Outputs
(shifted right)
Batch size
Seq. length
Embedded
tensor
d_model
Embedded
tensor
<START>
나는
널
사랑해
[
['I', 'love', 'you'],
['Good', 'job']
]
[
['<START>', '나는', '널', '사랑해'],
['<START>', '잘', '했어']
]

Positional Encoding

- Input Embedding에 더해저 위치 정보를 제공
- 길이 120짜리 문장
  - In a self-attention layer all of the keys, values and queries come from the same place, in this case, the output of the previous layer in the encoder…
- 임베딩 차원: 500

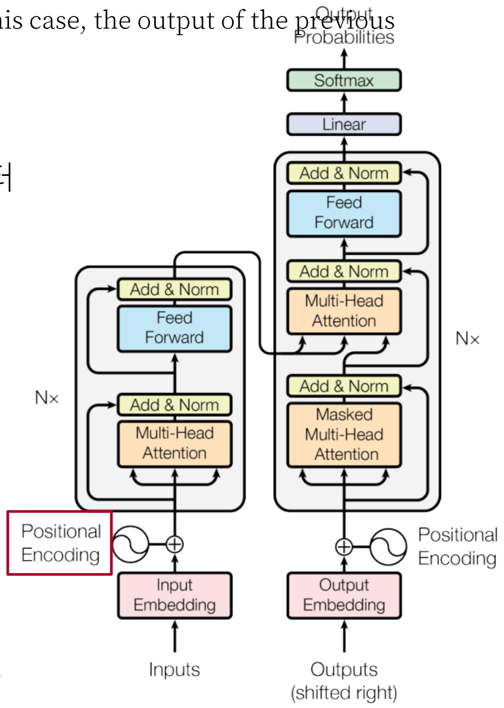
$$PE_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

첫번째 토큰의 위치를 나타내는 길이 500짜리 벡터



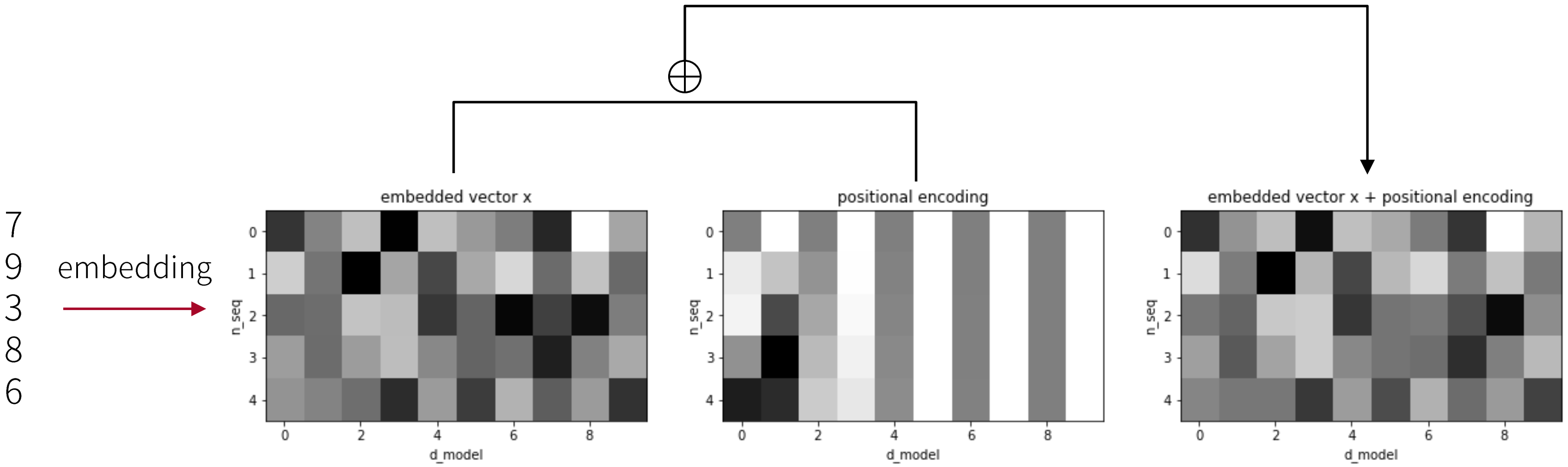
$$PE_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{model}}}}\right)$$

보통 이 방향으로 더 길게 PE를 만들
어 놓고 샘플 시퀀스만큼 잘라서 사용



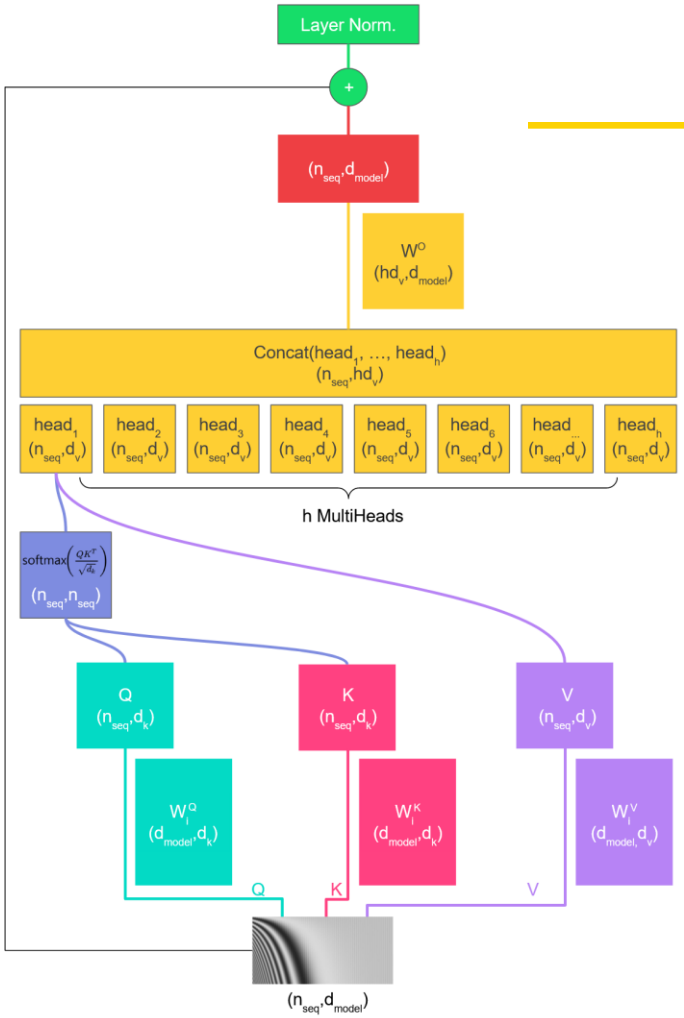
Positional Encoding Test

- 토큰 다섯 개짜리 가상 문장, i love you so much
  - 토큰 번호: [7, 9, 3, 8, 6]



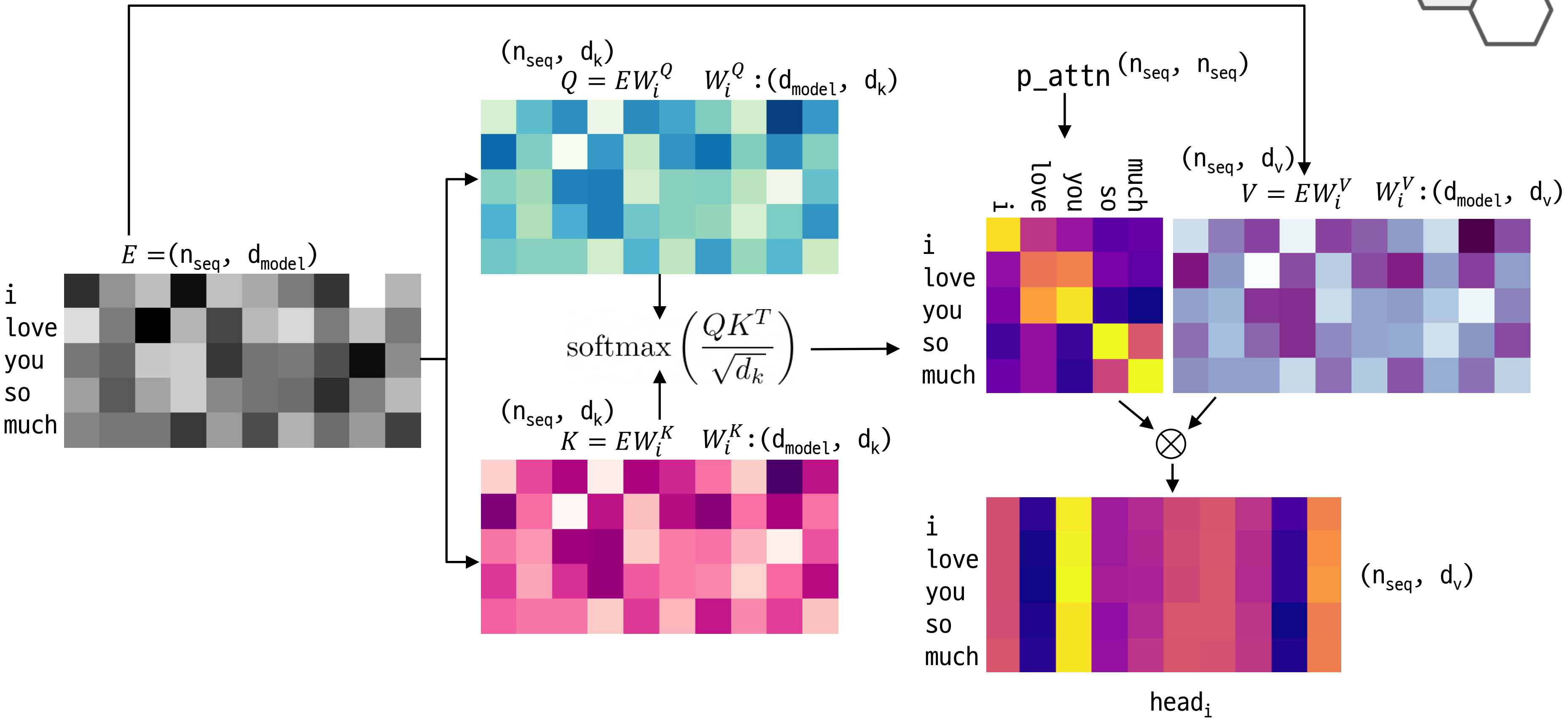
Encoder:Attention

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




```python
def attention(query, key, value, mask=None, dropout=None):
    "Compute 'Scaled Dot Product Attention'"
    # query: (nbatches, h, n_seq, d_k)
    # key:   (nbatches, h, n_seq, d_k)
    # value: (nbatches, h, n_seq, d_v) 인데 d_k=d_v로 두었음

    d_k = query.size(-1)

    scores = torch.matmul(query, key.transpose(-2, -1)) / math.sqrt(d_k)
    # scores: (nbatches, h, n_seq, n_seq)

    if mask is not None:
        scores = scores.masked_fill(mask == 0, -1e9)

    p_attn = F.softmax(scores, dim = -1)

    if dropout is not None:
        p_attn = dropout(p_attn)

    # torch.matmul(p_attn, value): (nbatches, h, n_seq, n_seq)*(nbatches, h, n_seq, d_v)
    # = (nbatches, h, n_seq, d_v),      p_attn: (nbatches, h, n_seq, nseq)
    return torch.matmul(p_attn, value), p_attn
```

Self-Attention



Recap Matrix Multiplication

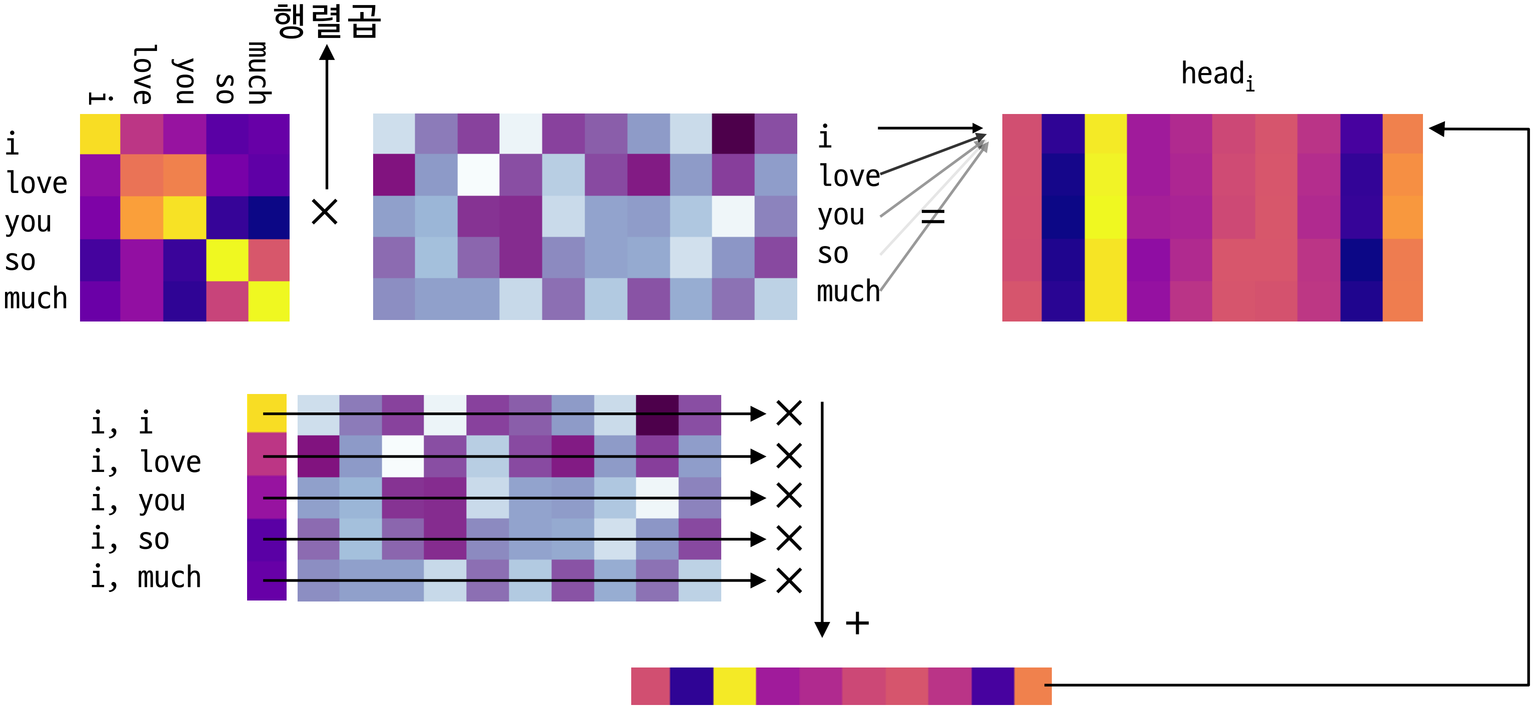
```
A = [[1, 2, 3],   B = [[10, 10, 10, 10, 10],   A*B = [[140, 140, 140, 140, 140],
     [3, 2, 1],        [20, 20, 20, 20, 20],          [100, 100, 100, 100, 100],
     [1, 3, 2]]        [30, 30, 30, 30, 30]]          [130, 130, 130, 130, 130]]
```

```
1 * [10, 10, 10, 10, 10] +
2 * [20, 20, 20, 20, 20] +
3 * [30, 30, 30, 30, 30] =

[140, 140, 140, 140, 140]
```

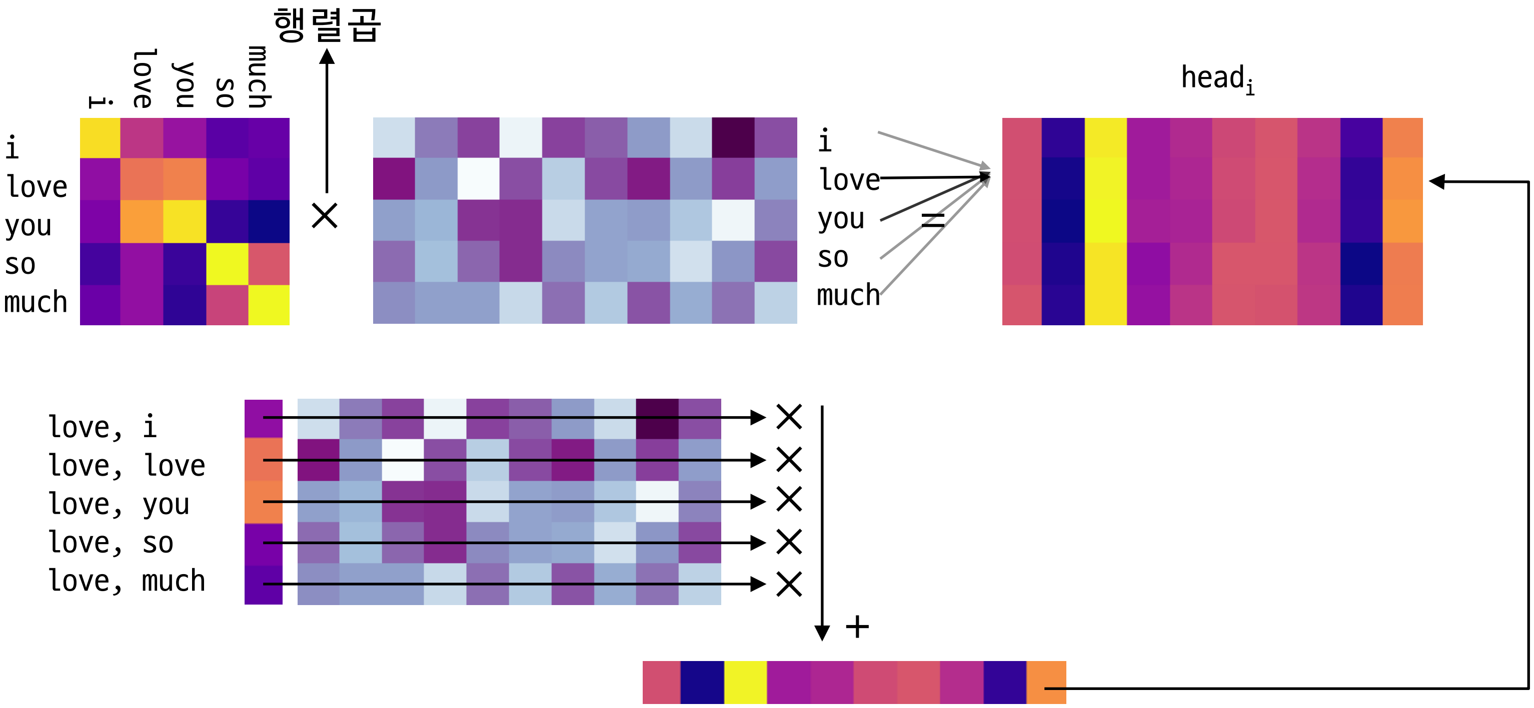
```
3 * [10, 10, 10, 10, 10] +
2 * [20, 20, 20, 20, 20] +
1 * [30, 30, 30, 30, 30] =

[100, 100, 100, 100, 100]
```

```
1 * [10, 10, 10, 10, 10] +
3 * [20, 20, 20, 20, 20] +
2 * [30, 30, 30, 30, 30] =

[130, 130, 130, 130, 130]
```

Self-Attention

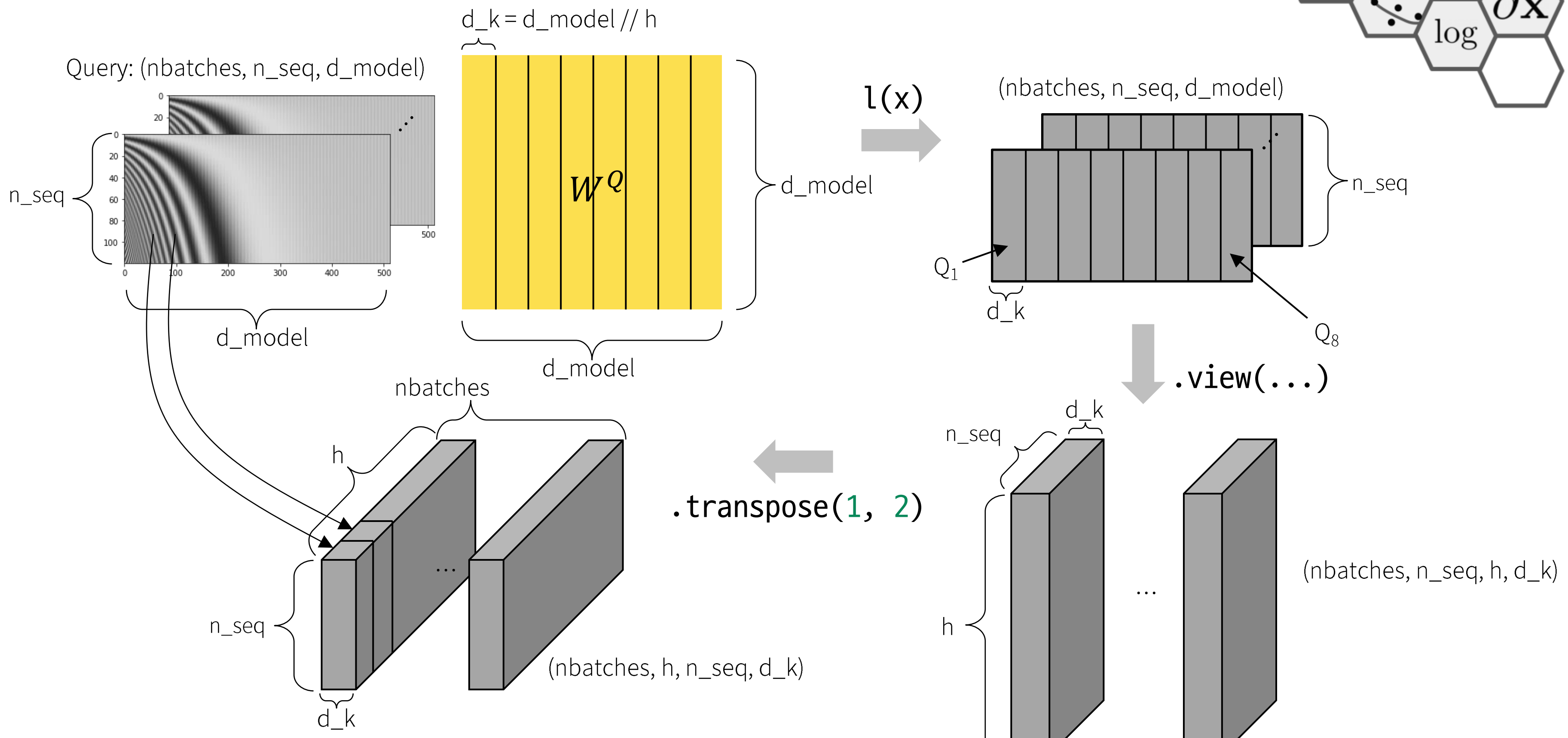


Self-Attention



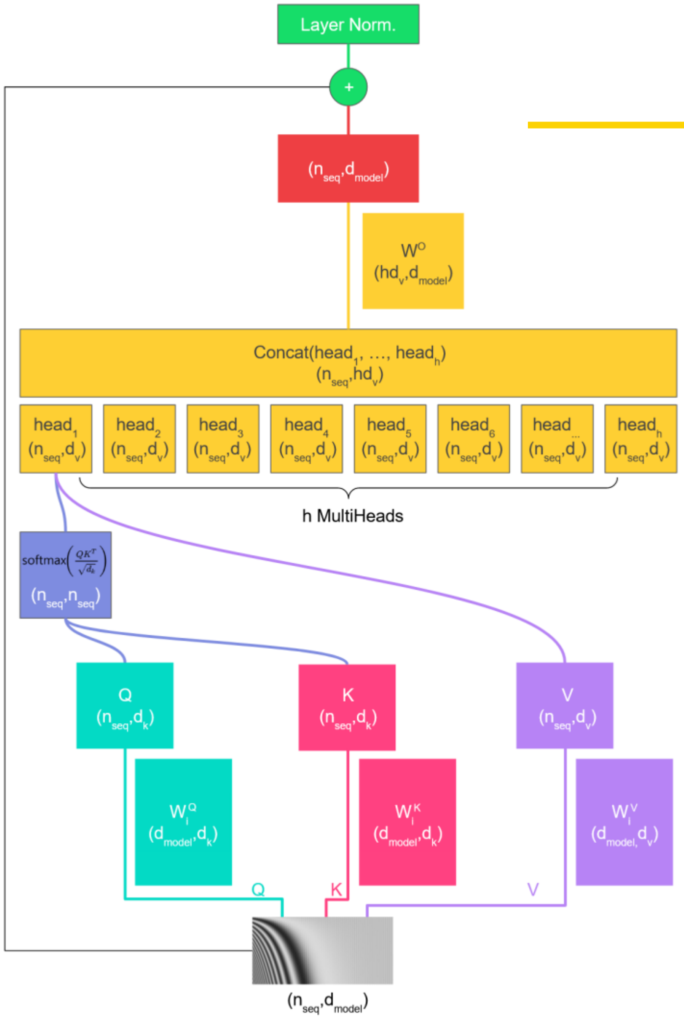
Multi-Head

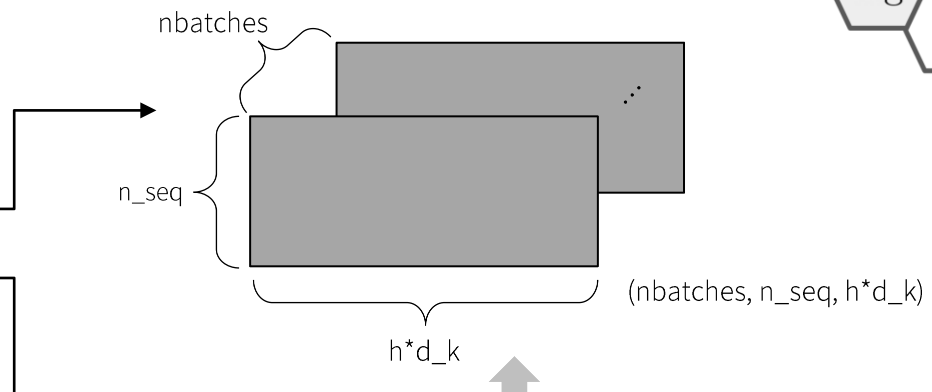
```python
query, key, value = \
    [l(x).view(nbatches, -1, self.h, self.d_k).transpose(1, 2)
     for l, x in zip(self.linears, (query, key, value))]
```



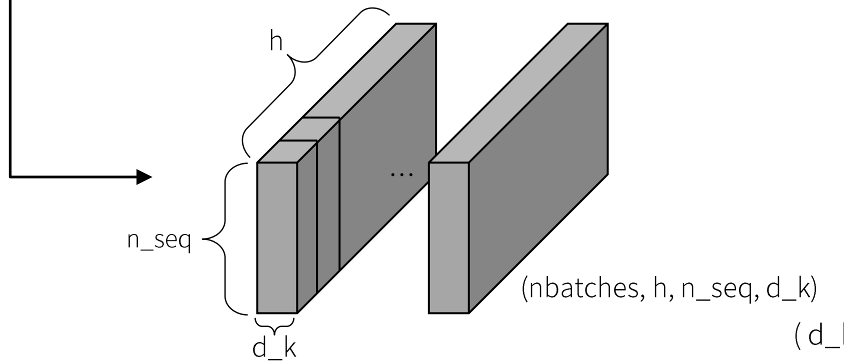
행렬곱 한번으로 (n_seq, d_k) 모양을 가지는 쿼리 h개 계산

Muti-Head Attention



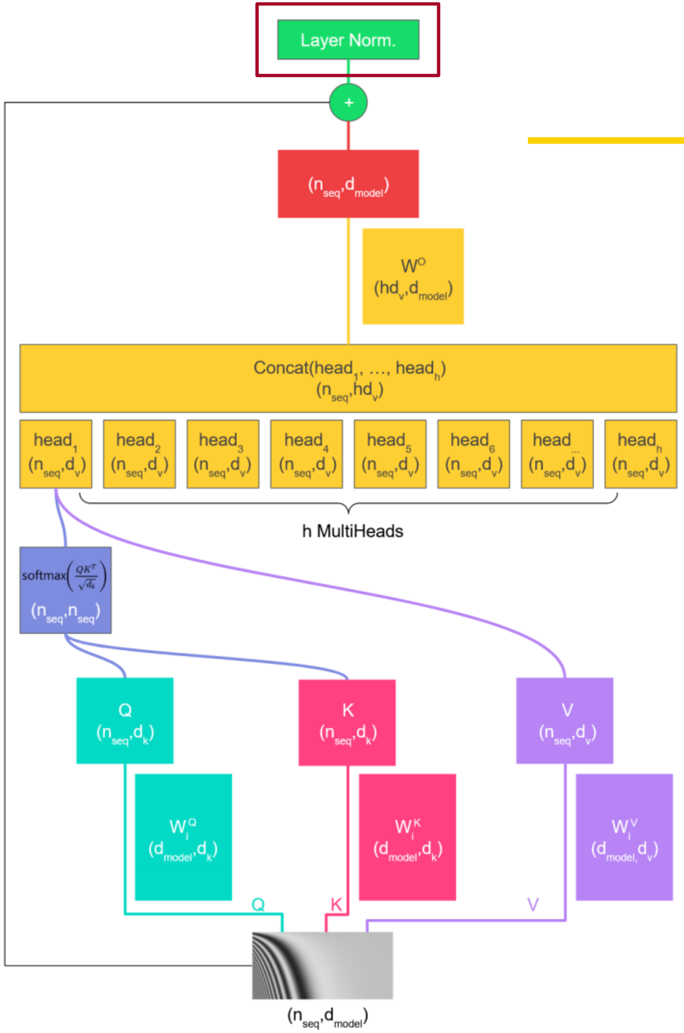


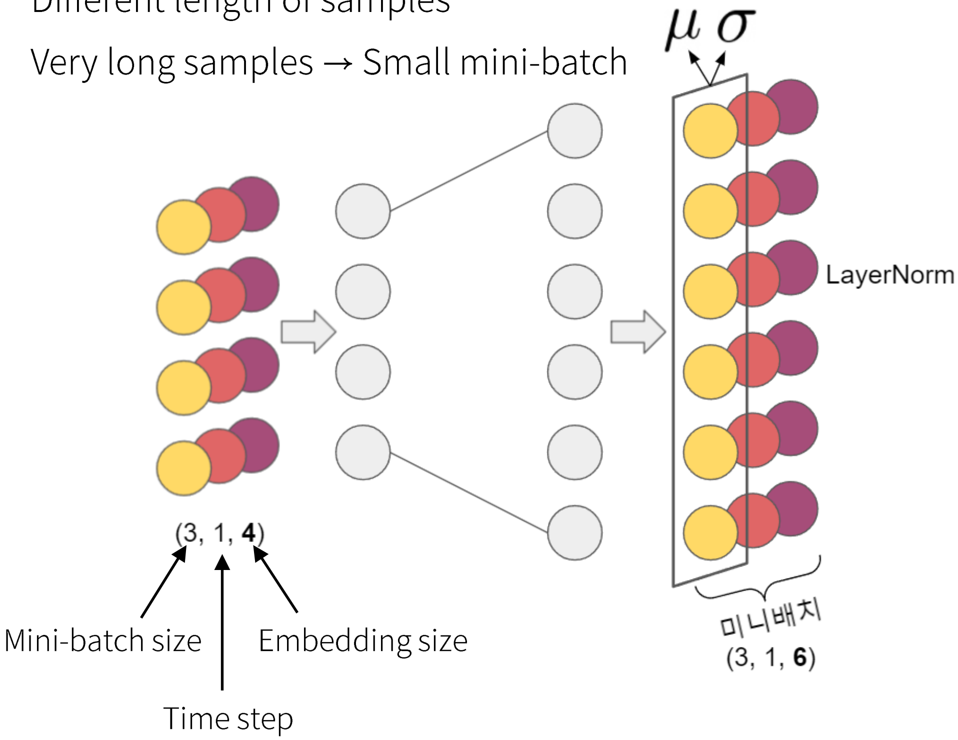

```python
x = x.transpose(1, 2).contiguous() \
                 .view(nbatches, -1, self.h * self.d_k)
```



Layer Normalization

- Batch Norm. 부적합
  - Different length of samples
  - Very long samples → Small mini-batch





Batch Normalization 1D

- Batch Normalization
  - 미니배치내 샘플과 feature를 이루는 모든 차원에 대해서 노멀라이즈
  - 주로 이미지 처리에 사용, CNN Layer
  - `bnorm = nn.BatchNorm1d(num_features=3)`

```
tensor([
  [[-1.1258, -1.1524, -0.2506, -0.4339],   야호 ← feature
   [-1.2904, -0.7911, -0.0209, -0.7185],   <PAD> ← feature
   [-1.2904, -0.7911, -0.0209, -0.7185]],  <PAD> ← feature

  [[ 0.1198,  1.2377,  1.1168, -0.2473],   와
   [-1.3527, -1.6959,  0.5667,  0.7935],   피곤해
   [ 0.4397,  0.1124,  0.6408,  0.4412]],  죽겠다

  [[-0.2159, -0.7425,  0.5627,  0.2596],   좋은
   [ 0.5229,  2.3022, -1.4689, -1.5867],   아침
   [-1.2904, -0.7911, -0.0209, -0.7185]]   <PAD>
])
(3, 3, 4)
```

Feature를 이루는 차원에 대해서 합산 →

샘플에 대해 합산 ↓

```
[-1.1258, -1.1524, -0.2506, -0.4339]
[ 0.1198,  1.2377,  1.1168, -0.2473] => -0.0726
[-0.2159, -0.7425,  0.5627,  0.2596]

[-1.2904, -0.7911, -0.0209, -0.7185]
[-1.3527, -1.6959,  0.5667,  0.7935] => -0.3950
[ 0.5229,  2.3022, -1.4689, -1.5867]

[-1.2904, -0.7911, -0.0209, -0.7185]
[ 0.4397,  0.1124,  0.6408,  0.4412] => -0.3340
[-1.2904, -0.7911, -0.0209, -0.7185]
```

Layer Normalization

- Layer Normalization
  - normalized_shape에 지정된 shape에 대해서만 노멀라이즈
  - 토큰별로 노멀라이즈, 자연어 처리에 사용, Linear Layer
  - lnorm = nn.LayerNorm(normalized_shape=4), normalized_shape은 입력의 마지막 차원부터 매칭

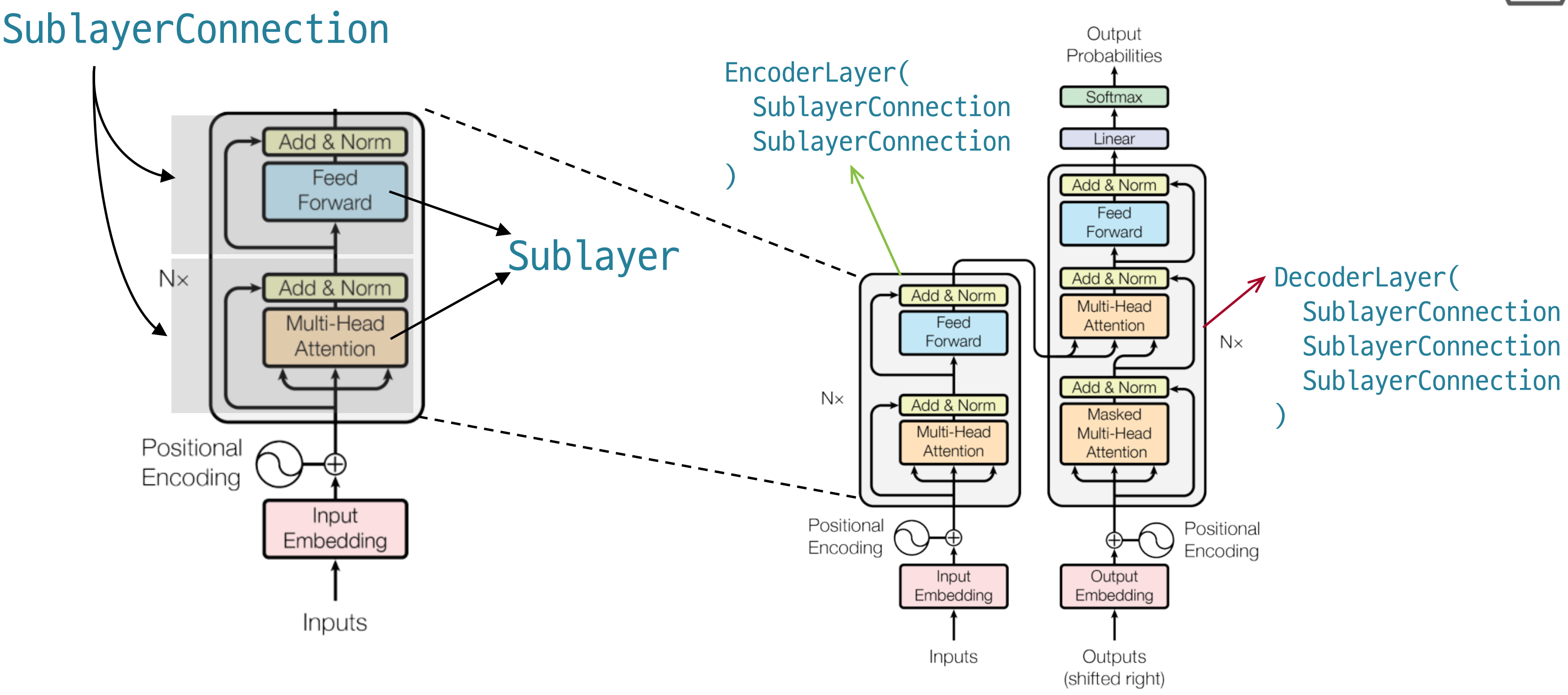
```
tensor([
  [[-1.1258, -1.1524, -0.2506, -0.4339],  야호  ← feature
   [-1.2904, -0.7911, -0.0209, -0.7185],  <PAD> ← feature
   [-1.2904, -0.7911, -0.0209, -0.7185]], <PAD> ← feature

  [[ 0.1198,  1.2377,  1.1168, -0.2473],  와
   [-1.3527, -1.6959,  0.5667,  0.7935],  피곤해
   [ 0.4397,  0.1124,  0.6408,  0.4412]], 죽겠다

  [[-0.2159, -0.7425,  0.5627,  0.2596],  좋은
   [ 0.5229,  2.3022, -1.4689, -1.5867],  아침
   [-1.2904, -0.7911, -0.0209, -0.7185]]  <PAD>
]) ────────────────────────────────→
     d_model
(3,3,4)
```

normalized_shape = d_model →

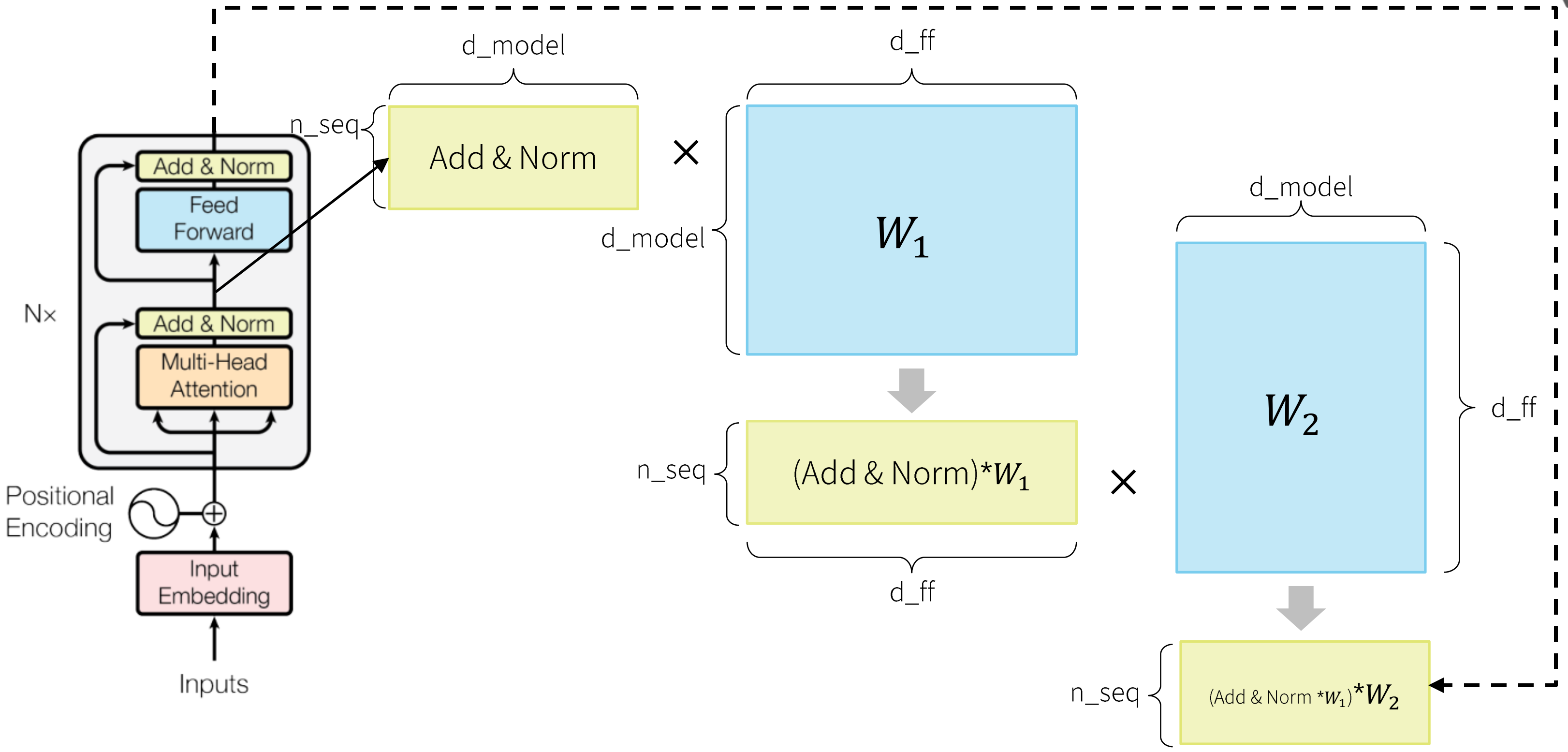
샘플, feature에 대해 합산 X ↓

```
[-1.1258, -1.1524, -0.2506, -0.4339] => -0.7407
[-1.2904, -0.7911, -0.0209, -0.7185] => -0.7052
[-1.2904, -0.7911, -0.0209, -0.7185] => -0.7052

[ 0.1198,  1.2377,  1.1168, -0.2473] =>  0.5567
[-1.3527, -1.6959,  0.5667,  0.7935] => -0.4221
[ 0.4397,  0.1124,  0.6408,  0.4412] =>  0.4085

[-0.2159, -0.7425,  0.5627,  0.2596] => -0.0340
[ 0.5229,  2.3022, -1.4689, -1.5867] => -0.0576
[-1.2904, -0.7911, -0.0209, -0.7185] => -0.7052
```

SublayerConnection



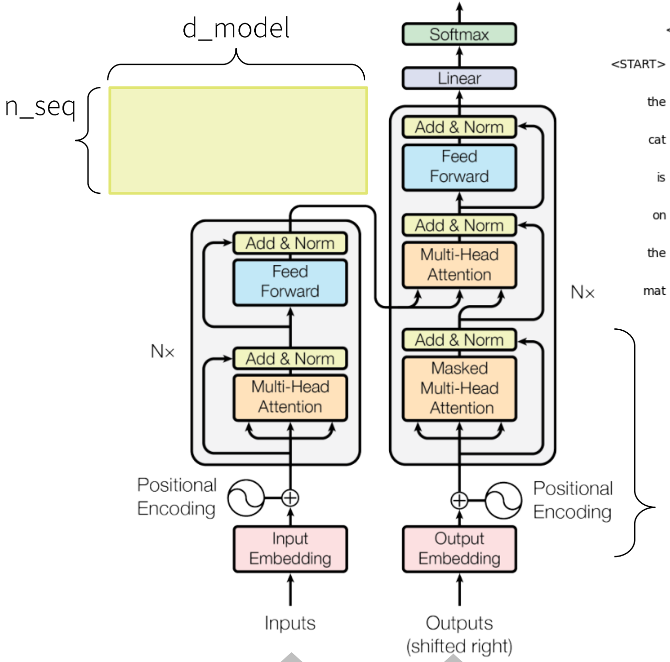
Position-wise Feed Forward

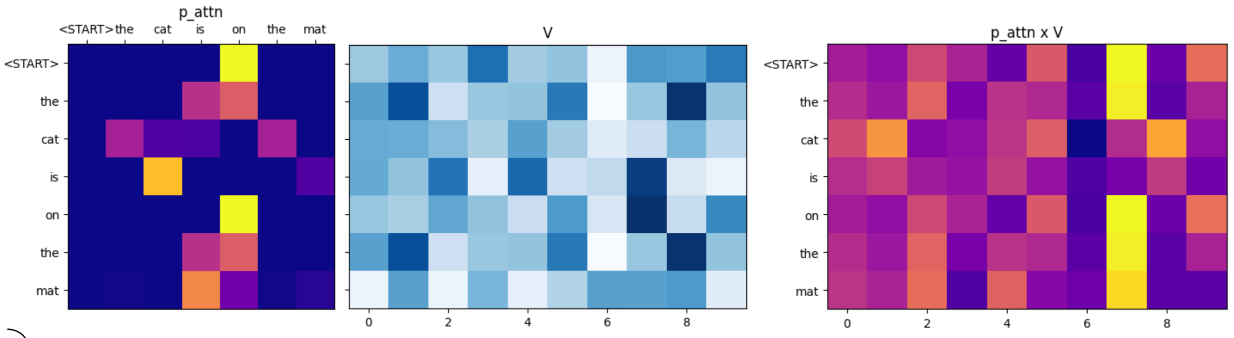


To decoder or next encoder layer

Decoder Self-Attention

the cat is on the mat <END>





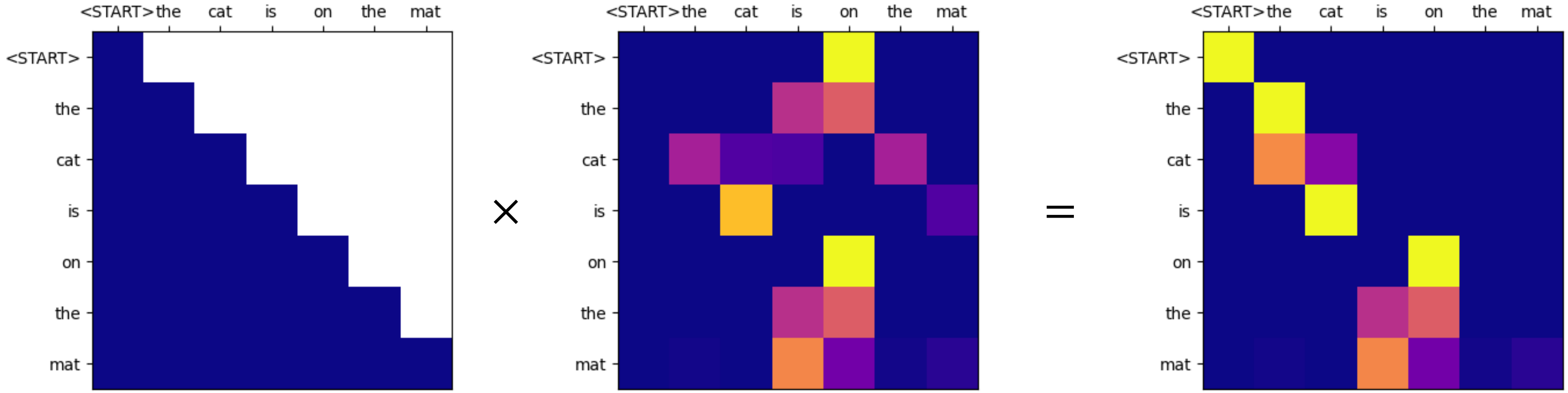
Encoder와 똑같은 과정으로 시작
그런데…
cat 이란 토큰은 is, on, the, mat에는 어텐션하면 안됨!

Le chat est sur le tapis

<START> the cat is on the mat

Decoder Mask

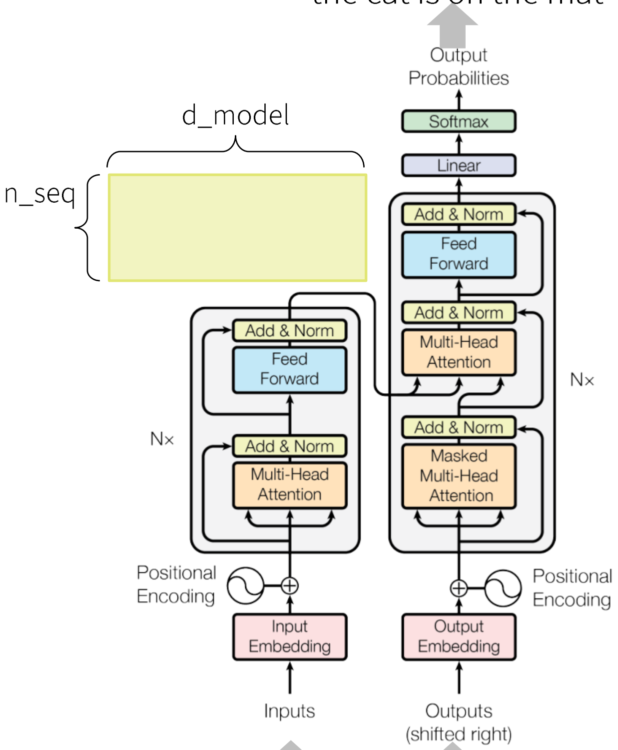
- subsequent mask
  - 디코더 입력에 대한 셀프 어텐션에서 n번째 단어가 n+1, n+2, … 번째 토큰에 어텐션 되지 않게 하는 마스크
  - 아래 cat이라는 토큰에 대한 임베딩에는 <START>, the 토큰 정보만 반영되야 함
  - Causal LM은 이전 단어 정보만을 사용해야 되기 때문
  - 아래 그림에서 masked p_attn을 사용하면 cat이라는 토큰은 자기 포함 자기 앞에 나온 토큰 <START>, the, cat만으로 구성됨

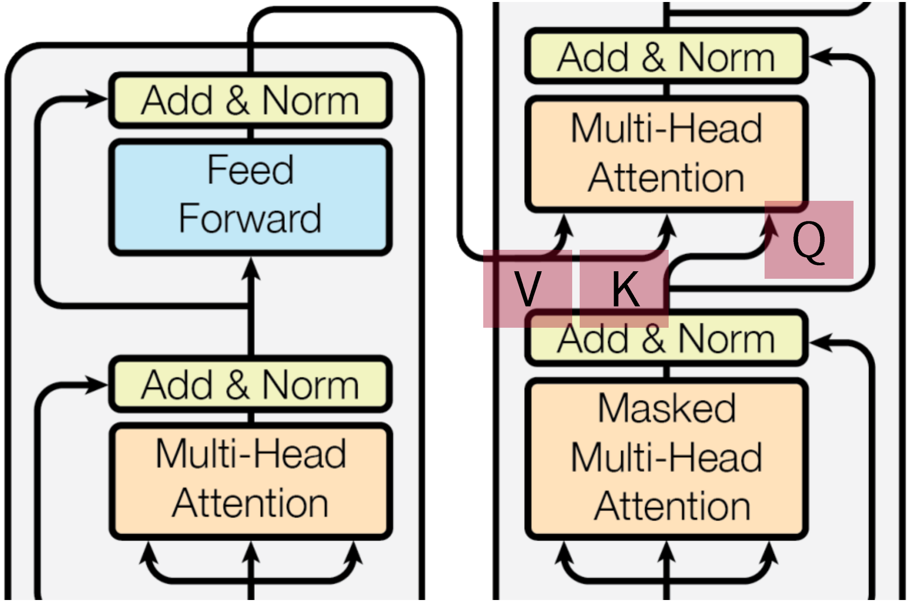


subsequent_mask　　　p_attn　　　masked p_attn

Cross Attention

the cat is on the mat <END>





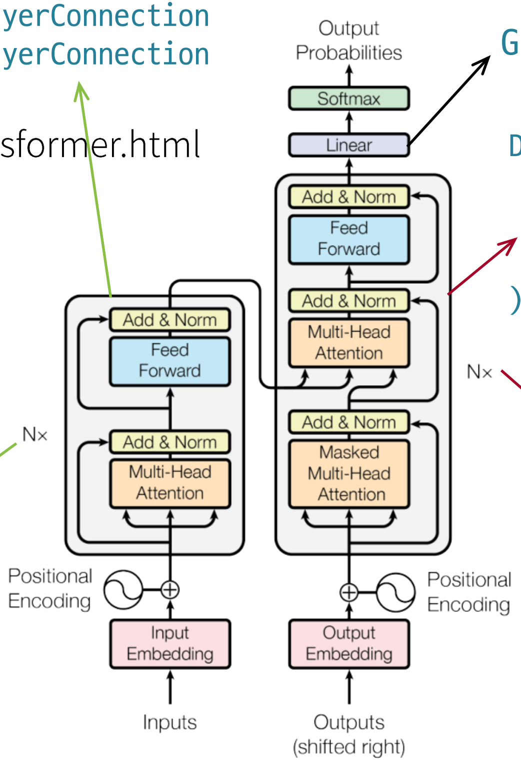
Le chat est sur le tapis

<START> the cat is on the mat

Transformer

- 전체 구현 및 코드 분석
  - 진짜로(?) 주석 달린 트랜스포머
  - https://metamath1.github.io/2021/11/11/transformer.html

```
EncoderLayer(
   SublayerConnection
   SublayerConnection
)
```



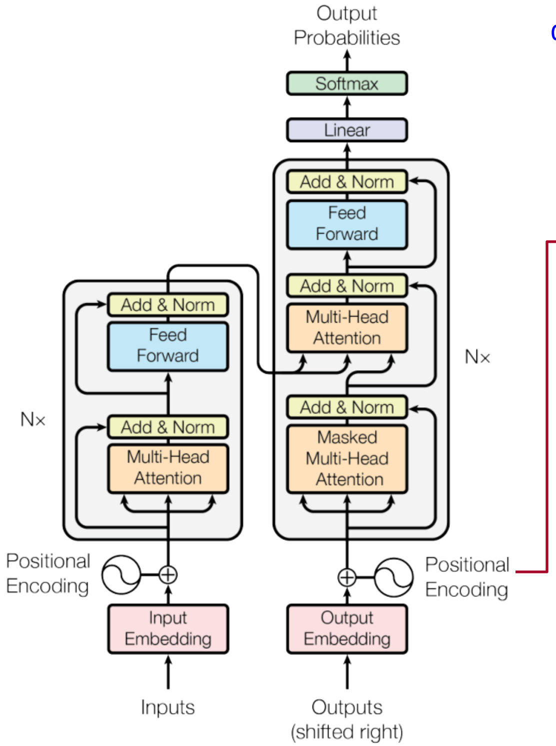
Generator

```
DecoderLayer(
   SublayerConnection
   SublayerConnection
   SublayerConnection
)
```

```
Encoder(
      EncoderLayer
      EncoderLayer
N=6   ...
      EncoderLayer
)
```

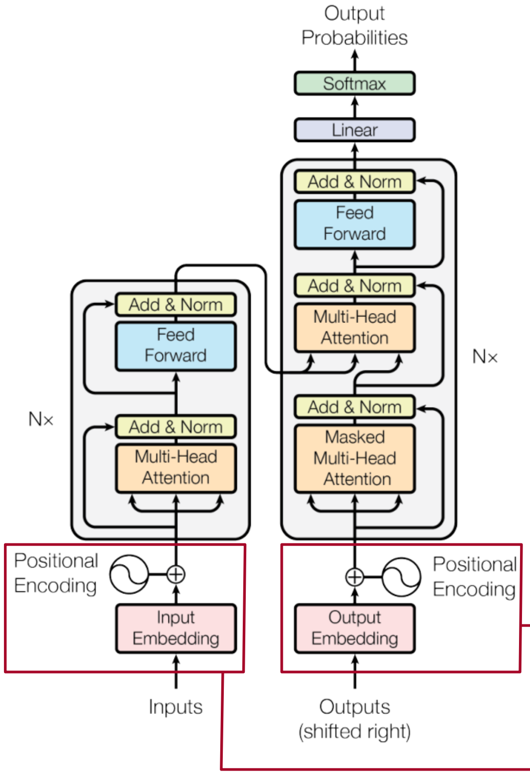
```
Decoder(
   Decoderlayer
   Decoderlayer
   ...             N=6
   Decoderlayer
)
```

Making Model




```python
def make_model(src_vocab, tgt_vocab, N=6,
               d_model=512, d_ff=2048, h=8, dropout=0.1):
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(

        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),

        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),

        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab)
    )

    return model
```

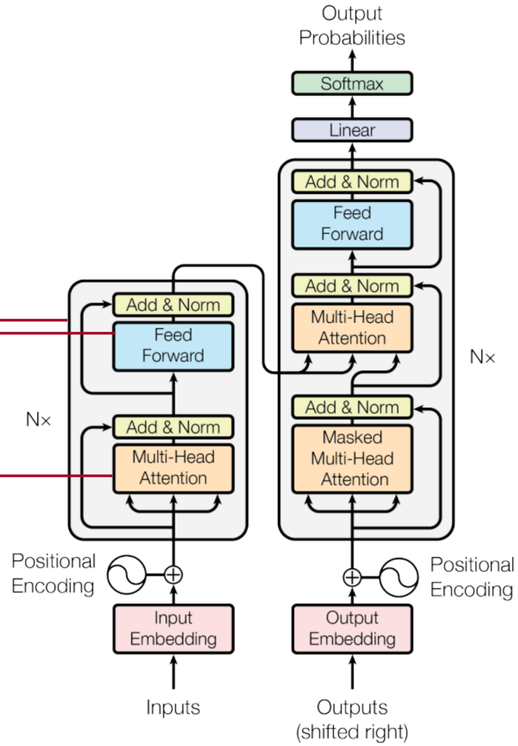
Making Model



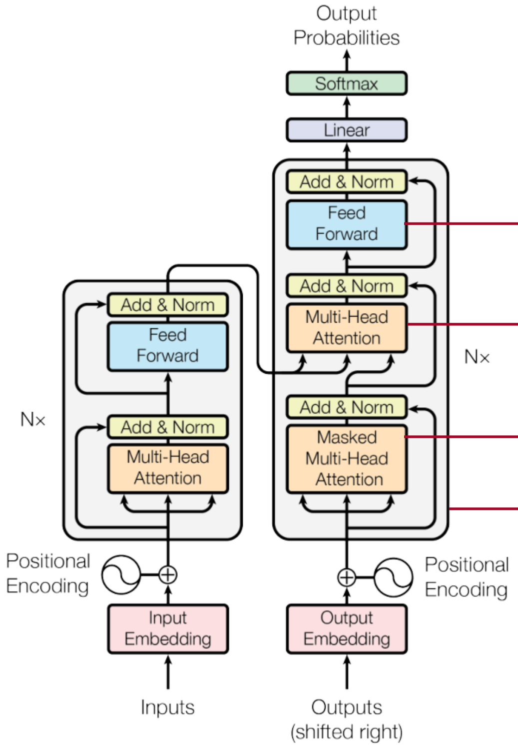

```python
def make_model(src_vocab, tgt_vocab, N=6,
               d_model=512, d_ff=2048, h=8, dropout=0.1):
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(

        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),

        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),

        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab)
    )

    return model
```

Making Model

```python
def make_model(src_vocab, tgt_vocab, N=6,
               d_model=512, d_ff=2048, h=8, dropout=0.1):
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(

        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),

        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),

        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab)
    )

    return model
```

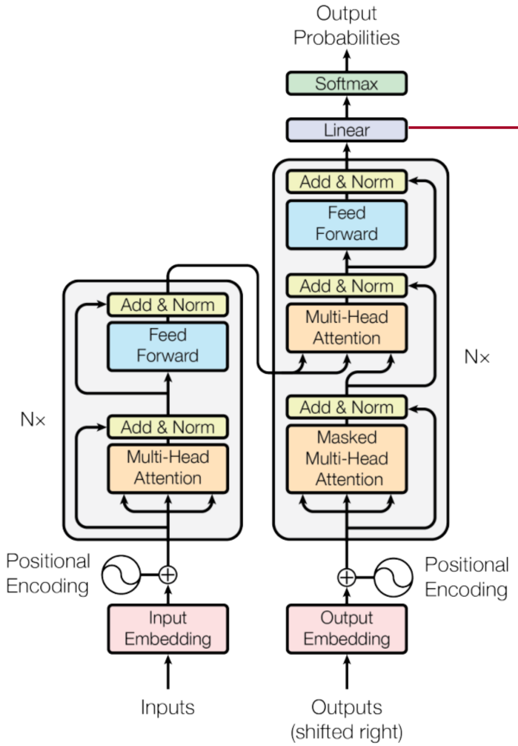


Making Model




```python
def make_model(src_vocab, tgt_vocab, N=6,
               d_model=512, d_ff=2048, h=8, dropout=0.1):
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(
        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),
        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),
        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab)
    )

    return model
```

Making Model




```python
def make_model(src_vocab, tgt_vocab, N=6,
               d_model=512, d_ff=2048, h=8, dropout=0.1):
    c = copy.deepcopy
    attn = MultiHeadedAttention(h, d_model)
    ff = PositionwiseFeedForward(d_model, d_ff, dropout)
    position = PositionalEncoding(d_model, dropout)
    model = EncoderDecoder(

        Encoder(EncoderLayer(d_model, c(attn), c(ff), dropout), N),

        Decoder(DecoderLayer(d_model, c(attn), c(attn), c(ff), dropout), N),

        nn.Sequential(Embeddings(d_model, src_vocab), c(position)),
        nn.Sequential(Embeddings(d_model, tgt_vocab), c(position)),
        Generator(d_model, tgt_vocab)
    )

    return model
```

Learning Late Schedule

- 다음 식으로 학습률 조정

얼마까지 증가시킬지 지정

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

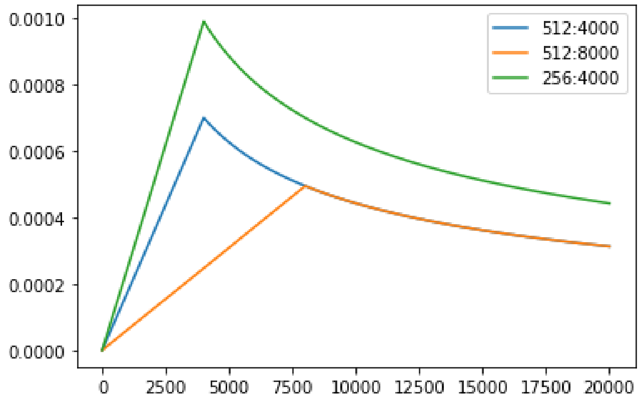
optimizer.step()마다 1 증가

- 학습 초기
  - step_num에 비례하여 선형적으로 증가

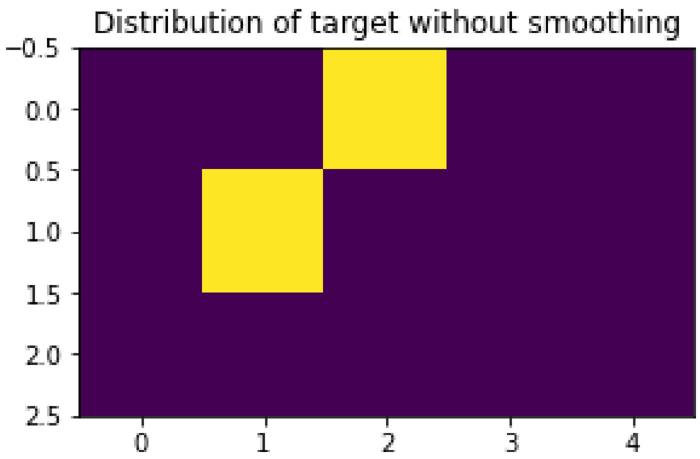
$$step\_num \cdot warmup\_steps^{-1.5} = \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$

- $step\_num = warmup\_steps$ 이후
  - step_num에 반비례하여 감소

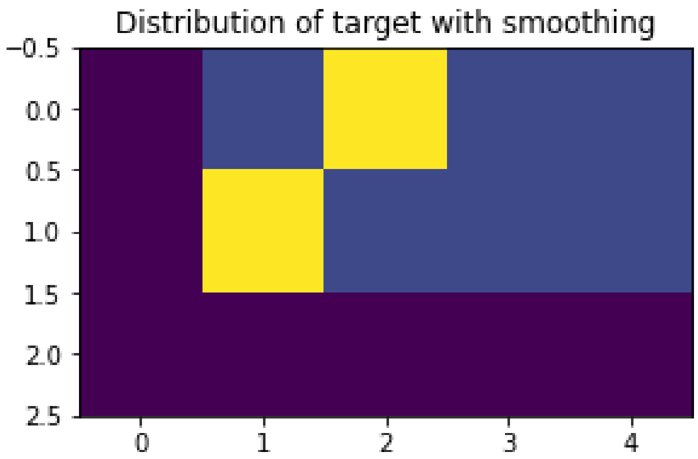
$$step\_num^{-0.5} = \min(step\_num^{-0.5}, step\_num \cdot warmup\_steps^{-1.5})$$



Label Smoothing



[PAD] [TKN1] [TKN2] [TKN3] [TKN4]



[PAD] [TKN1] [TKN2] [TKN3] [TKN4]

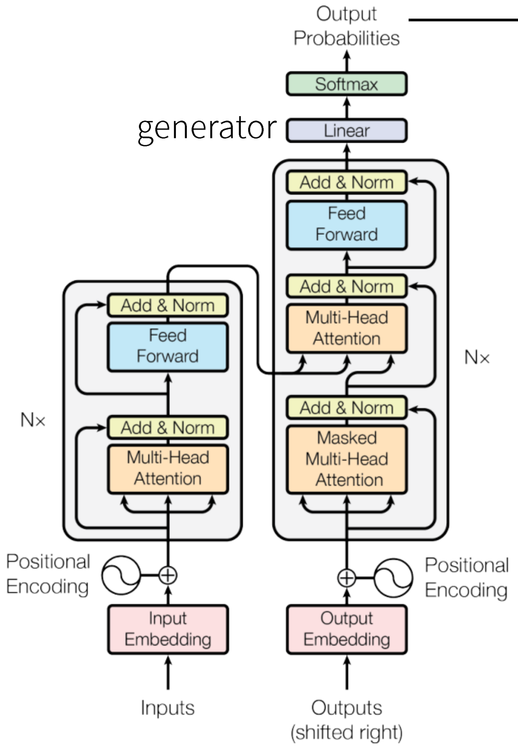
Target label

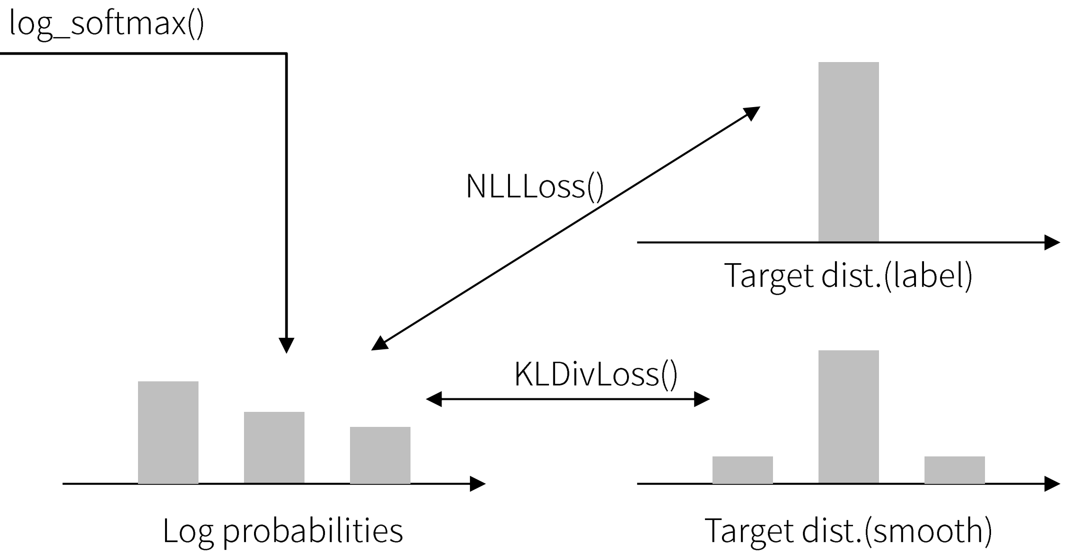
2

1

0

Loss





$$D_{\mathrm{KL}}(P||Q) = \sum_x P(x) \log \frac{P(x)}{Q(x)}$$

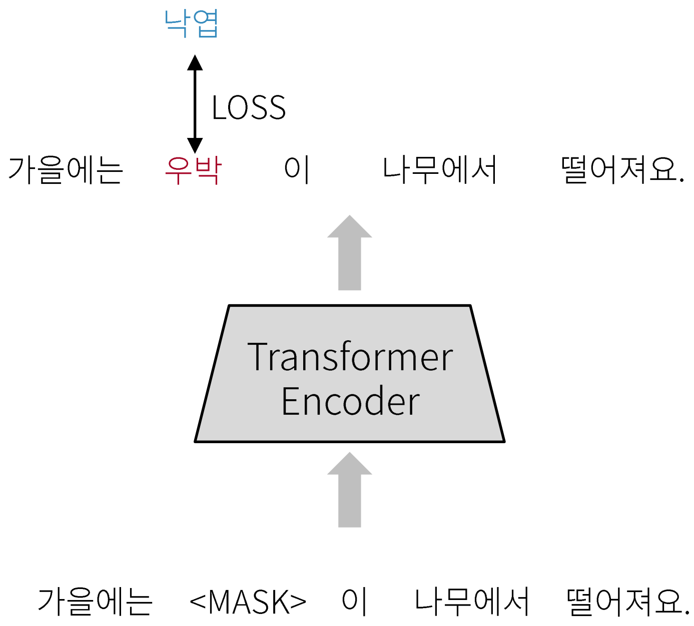
https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence

PLM

- PLM: Pre-trained Language Model
  - CNN과 마찬가지로 미리 학습된 거대 모델을 Down Stream Task에 Fine Tuning
  - 사전학습 Task에 대해 학습되어 언어에 대한 일반 이해 능력 향상
  - Self-supervised Learning 전략 사용
    - Masked Language Model
    - Causal Language Model
- Pre-trained Models
  - Elmo: 임베딩에 대한 선학습 모델, 고정 임베딩 벡터를 생성하지 않고 입력의 문맥에 따른 임베딩 벡터 생성, NLP에서도 사전학습이 성공적으로 사용될 수 있음을 최초로 보임
  - BERT: Masked Language Model, Transformer Encoder를 이용하여 NLU에 적합
  - GPT: Causal Language Model, Transformer Decoder를 이용하여 NLG에 적합

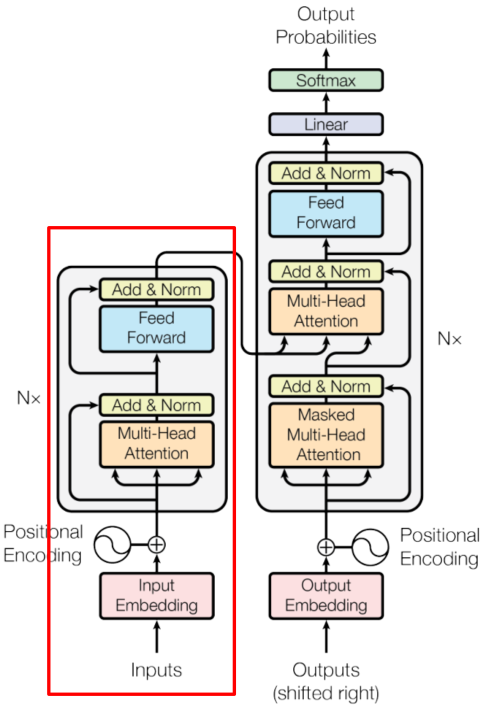
Masked LM

- 텍스트
  - 가을에는 낙엽이 나무에서 떨어져요.
  - 가을에는 ___이 나무에서 떨어져요.
- 문장의 전후 관계를 파악하는 능력을 배울 수 있음



Masked LM: BERT

- BERT: Bidirectional Encoder Representations from Transformers
- Transformer에서 Encoder 부분만 사용
- Google에서 개발



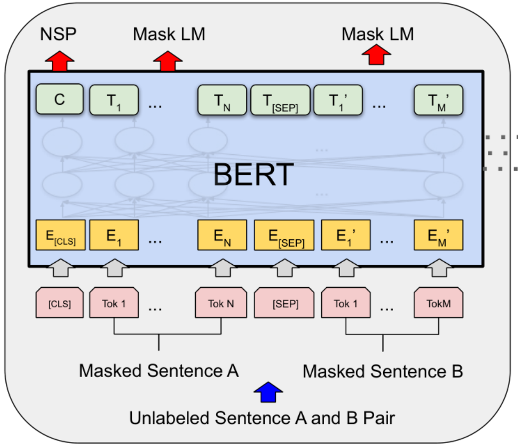
BERT

- BERT: Bidirectional Encoder Representations from Transformers
  - $\text{BERT}_{\text{BASE}}$: (L=12(# layer), H=768(hidden vector), A=12(Attn. head), Total Parameters=110M)
  - $\text{BERT}_{\text{LARGE}}$: (L=24, H=1024, A=16, Total Parameters=340M)
- Pre-training Data
  - the BooksCorpus (800M words)
  - English Wikipedia (2,500M words)
- Pre-training Task
  - MLM: Masked Language Model, 지운 단어 맞추기
  - NSP: Next Sentence Prediction, 두 문장이 이어지는 문장인지 맞추기

BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob D. et al., 2018

Pre-training

1	tkn1	tkn2	좀	tkn4	매우	tkn6	tkn7	tkn8	자연어	tkn10	tkn11	tkn12
loss			loss		loss				loss			
1	tkn1	tkn2	매우	tkn4	꽤	tkn6	tkn7	tkn8	자연어	tkn10	tkn11	tkn12

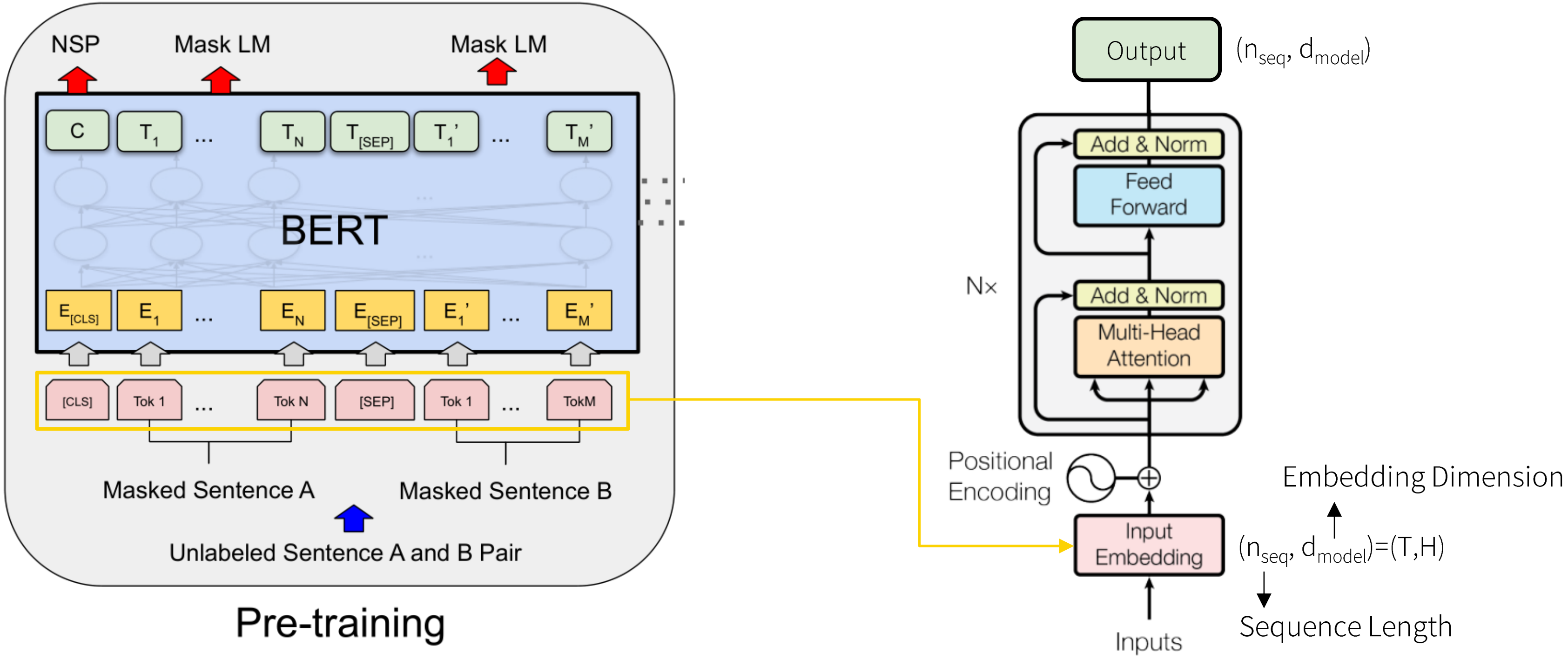


Pre-training

[CLS] 딥러닝 공부는 [MASK] 어렵지만 [MASK] 재미있다 [SEP] 트랜스포머는 [MASK] 처리 모델이다 [SEP]

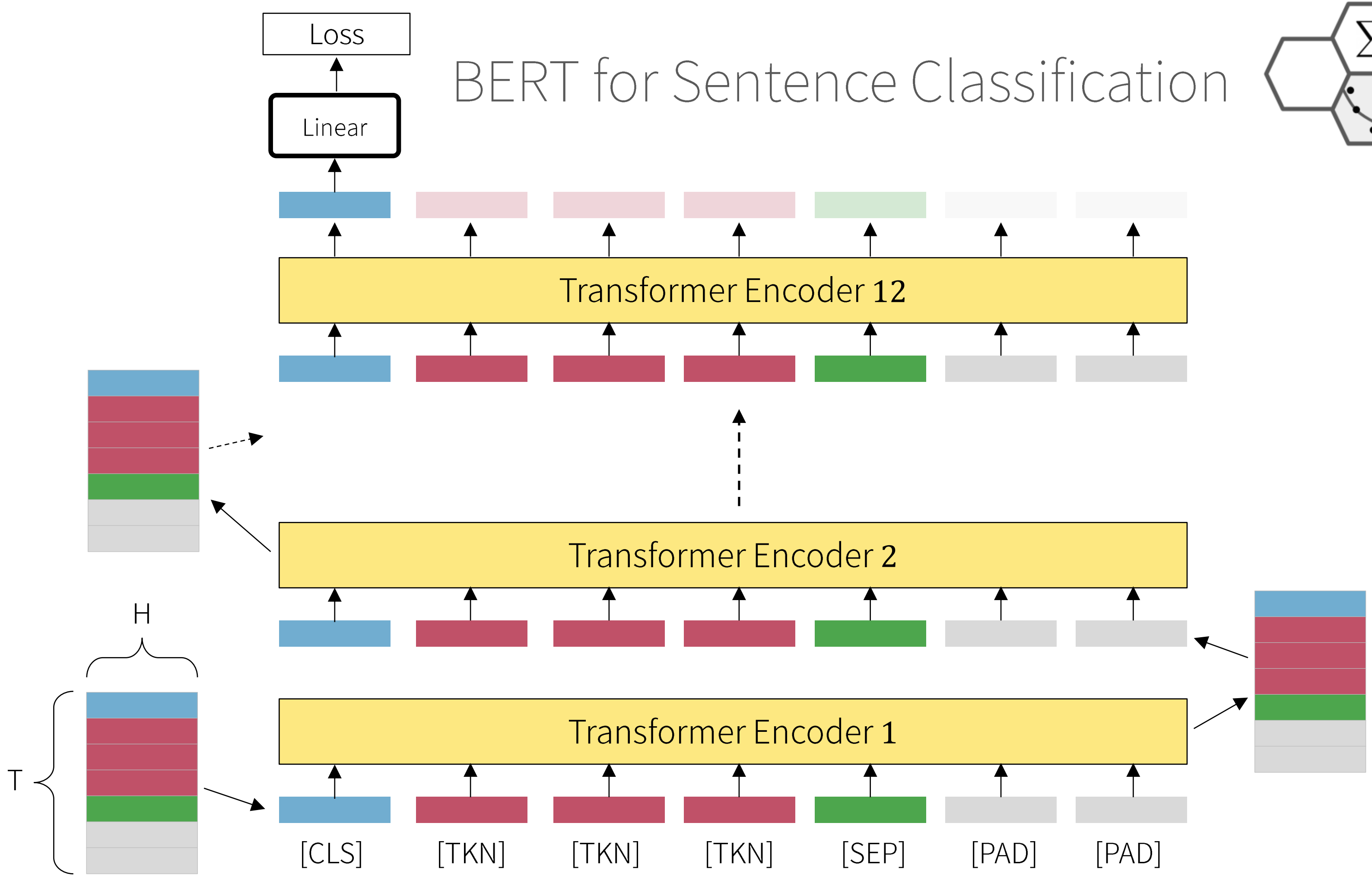
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob D. et al., 2018

Bert is Transformer Encoder



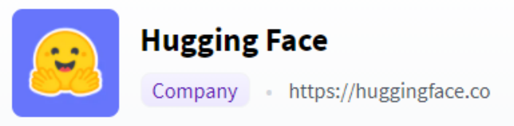
BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob D. et al., 2018

BERT for Sentence Classification

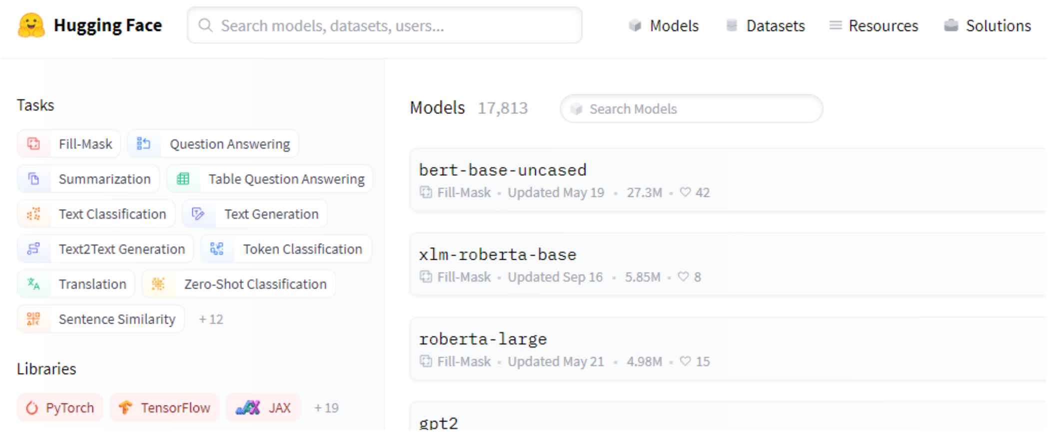


Hugging Face

- https://huggingface.co/

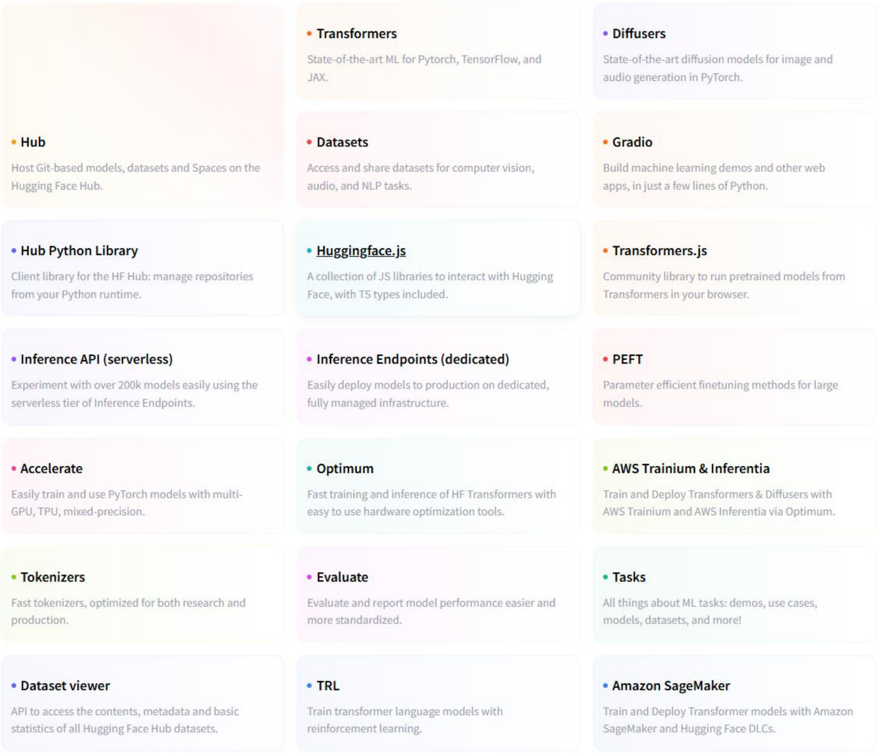


- Transformer 기반 라이브러리를 오픈소스로 제공
- 모델을 등록하고 다운받아 제공
- 다양한 선학습pretrained 모델 사용 가능



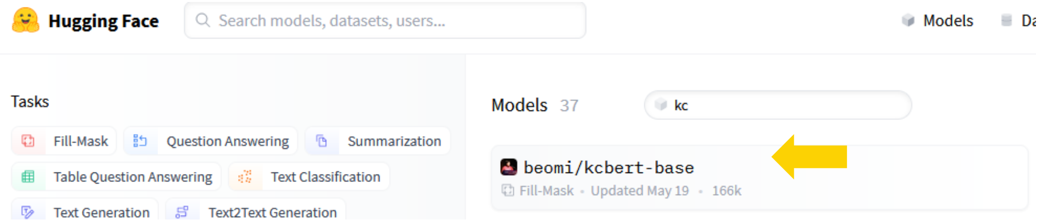
Hugging Face Library

- https://huggingface.co/docs
- **Transformers, Datasets, Tokenizers**
- Accelerate, PEFT, TRL, Bitsandbytes, Diffusers



BERT를 사용한 NSMC

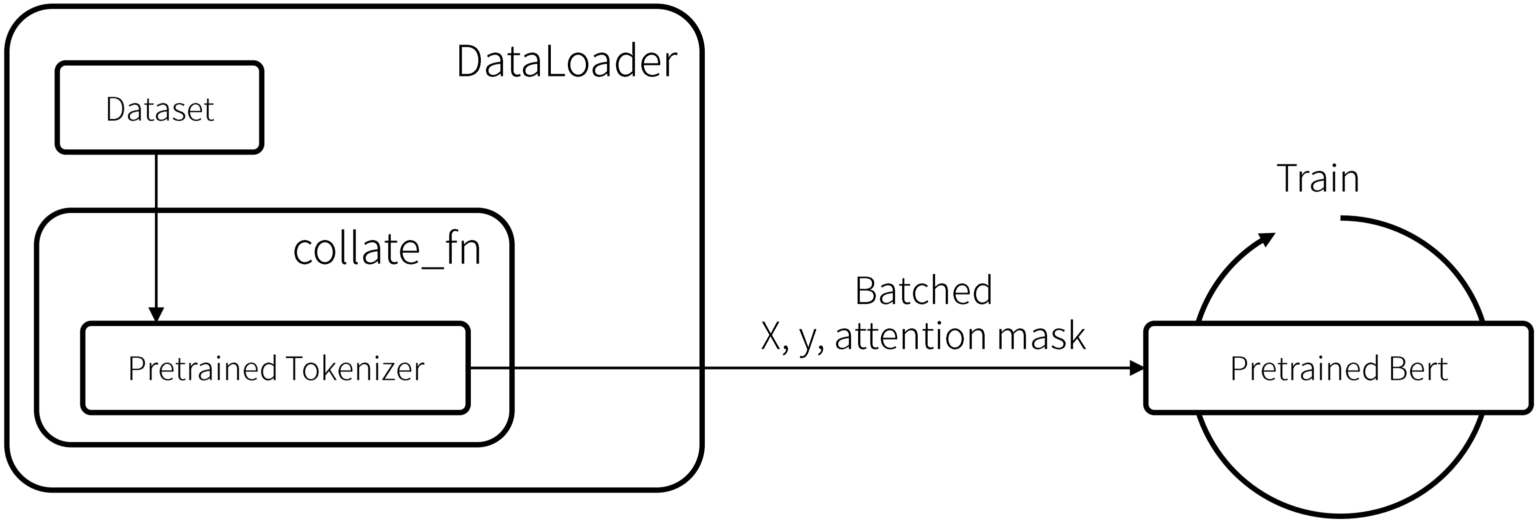
- 모델 다운로드
  - beomi/kcbert-base, beomi/kcbert-large
  - https://huggingface.co/models?sort=downloads&search=kc



- 전처리 최소화
- 모델에서 함께 제공하는 학습된 토크나이저
  - 네이버, 뉴스 사이트 등에서 모은 댓글 데이터
  - https://www.kaggle.com/junbumlee/kcbert-pretraining-corpus-korean-news-comments
- DataLoader
  - Custom collate_fn

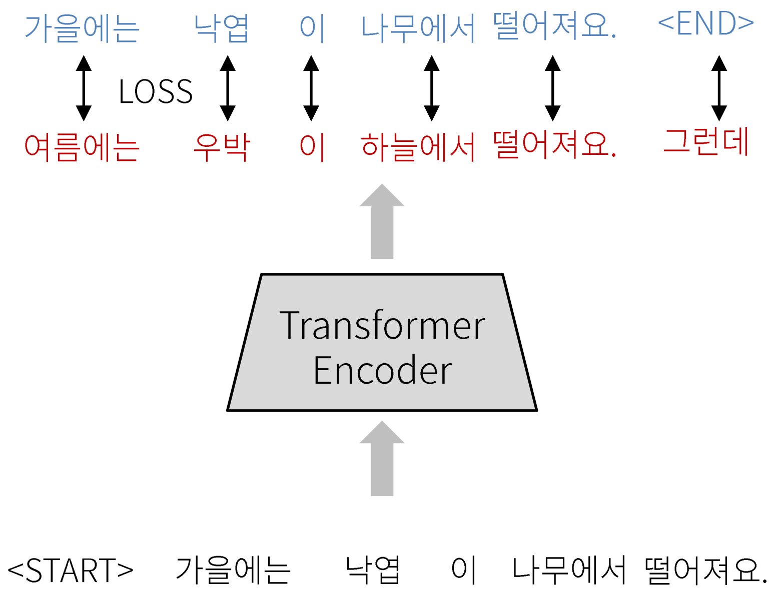
전체 프로세스

```
for i, data in enumerate(tqdm(train_loader)):
```



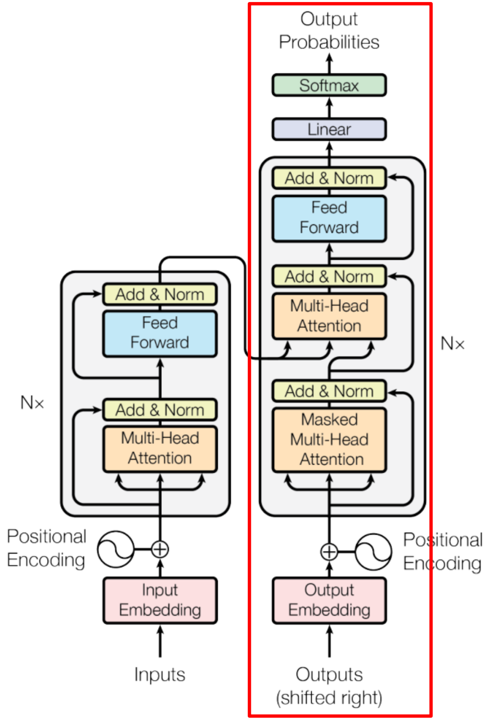
Causal LM

- 텍스트
  - <START>가을에는 낙엽이 나무에서 떨어져요.<END>
  - 입력: <START>가을에는 낙엽이 나무에서 떨어져요.
  - 타겟: 가을에는 낙엽이 나무에서 떨어져요.<END>
- 이전 단어(토큰)를 보고 다음 토큰을 예측하는 능력을 배울 수 있음



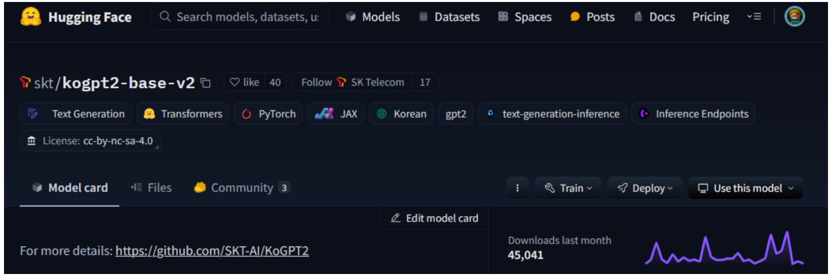
Causal LM: GPT2

- GPT: Generative Pre-Training
- Transformer에서 decoder 부분만 사용
- OpenAI에서 개발



GPT2를 사용한 기사 제목 생성

- 모델 다운로드
  - skt/kogpt2-base-v2
  - https://huggingface.co/skt/kogpt2-base-v2

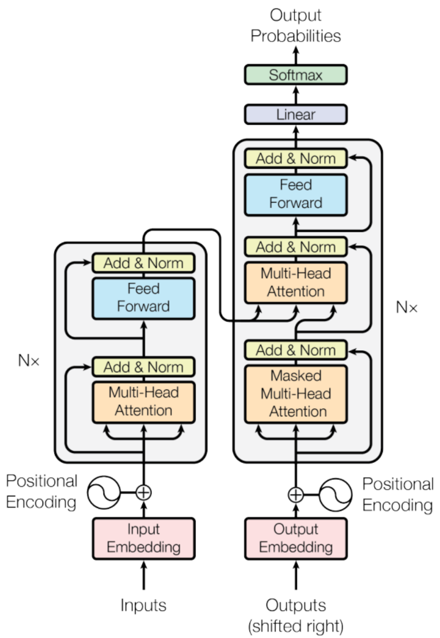


- gpt로 만든 가상의 뉴스 제목 데이터를 타겟으로 파인튜닝
- 파인튜닝 모델을 HuggingFace Hub로 업로드해서 활용

Seq2Seq Language Model

Teacher Forcing

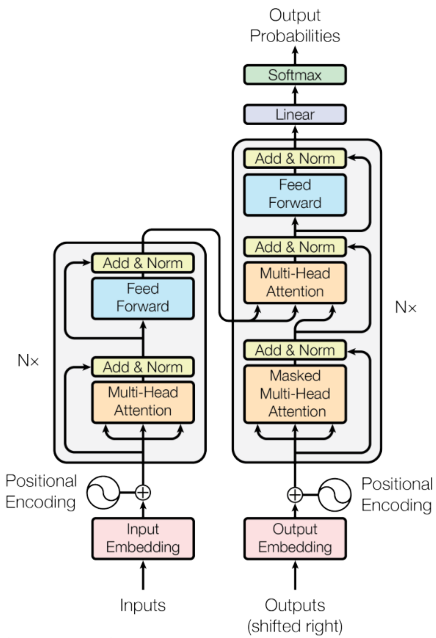
어떤 학문이든지 일정의 성취를 이루기 위해서는 끊임없는 반복이 필요하다</s>



Any academic achievement requires constant repetition. <pad> 어떤 학문이든지 일정의 성취를 이루기 위해서는 끊임없는 반복이 필요하다

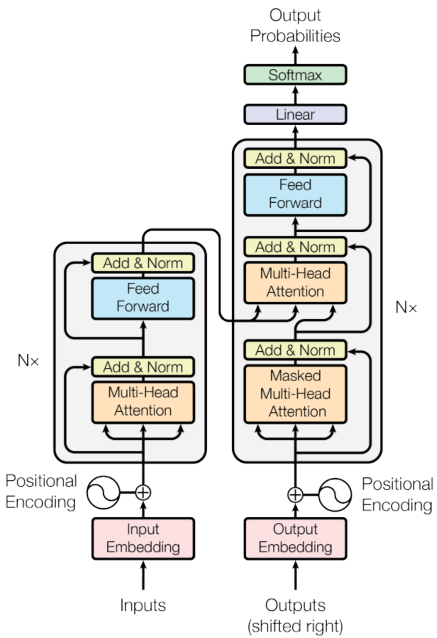
Neural Machine Translation

나는



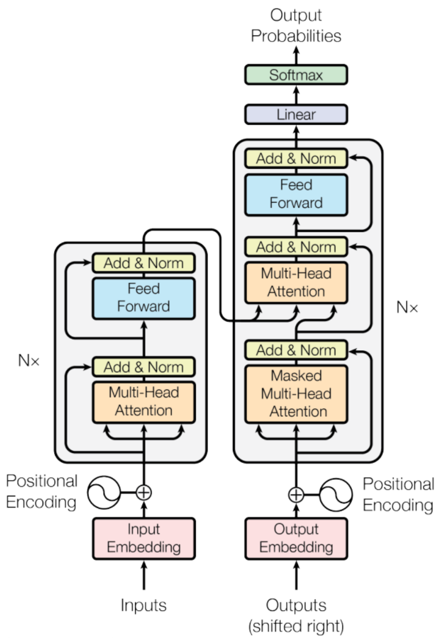
I live in Chicago. <pad>

시카고에



I live in Chicago. <pad> 나는

산다



I live in Chicago. <pad> 나는 시카고에

Seq2Seq Model: T5

- T5 모델 기반 한국어 번역기 튜토리얼
  - 한국어에 사전 학습된 T5 모델 활용

A Gentle Introduction to Creating an English-to-Korean translator with Transformers

AI | T5 | TRANSFORMERS | HUGGING FACE | 번역기 | 한국어 번역기

AUTHOR
Jo, Joonu (metamath@gmail.com)

PUBLISHED
February 27, 2023

친절한 영어-한국어 번역기 만들기 A Gentle Introduction to Creating an English-to-Korean translator with Transformers

https://metamath1.github.io/blog/posts/gentle-t5-trans/gentle_t5_trans.html