

# Transformer Based Dialog Engine for Workers

Younggi Lee

2022.01.12

# “ CATEGORY ”



Background



Transformer  
& code

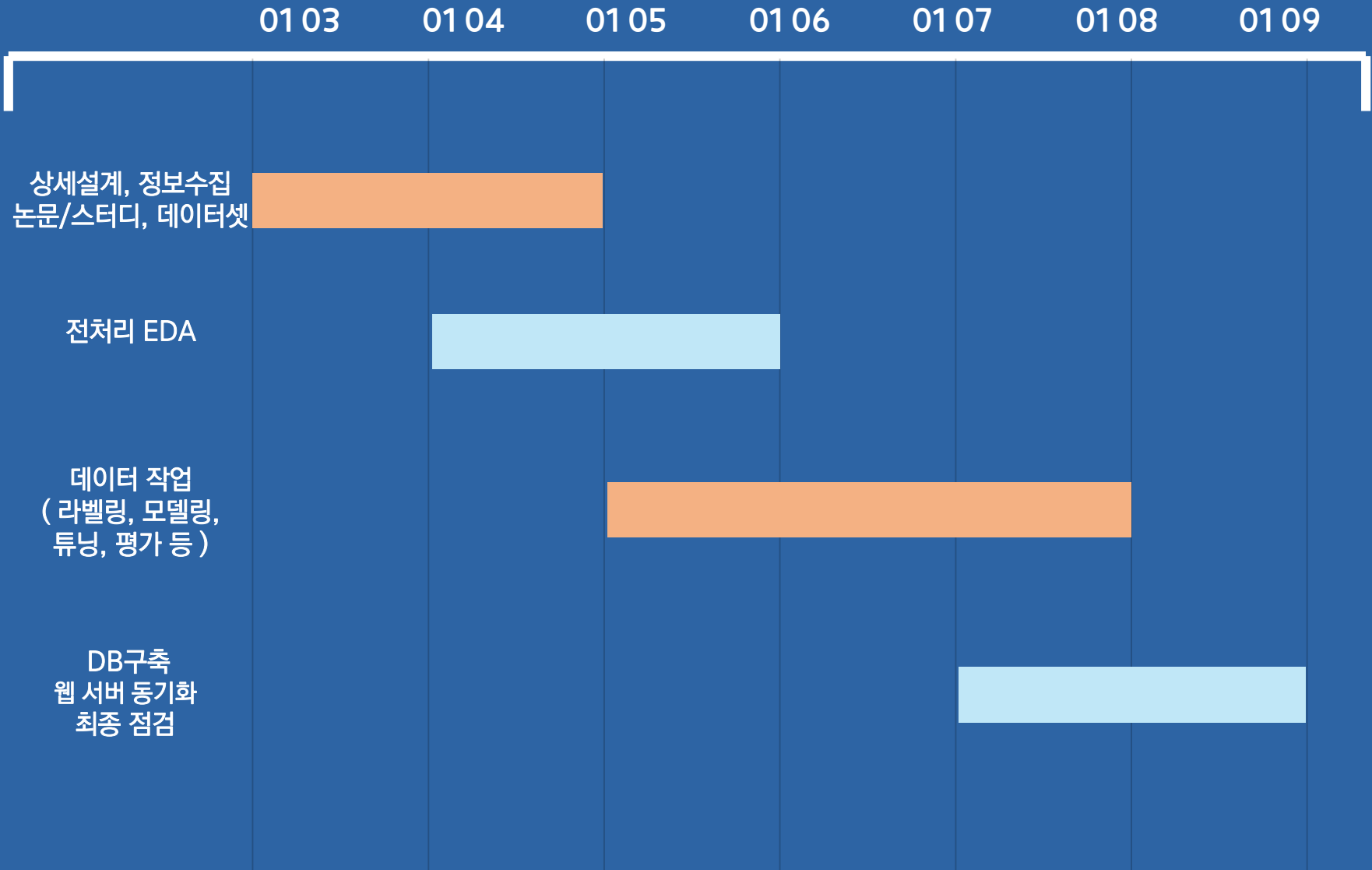


Result



Things to  
improve

# Time Table



☑ Chapter: 1

☐ Chapter: 2

☐ Chapter: 3

☐ Chapter: 4

Chapter



Background

주제 선정 배경

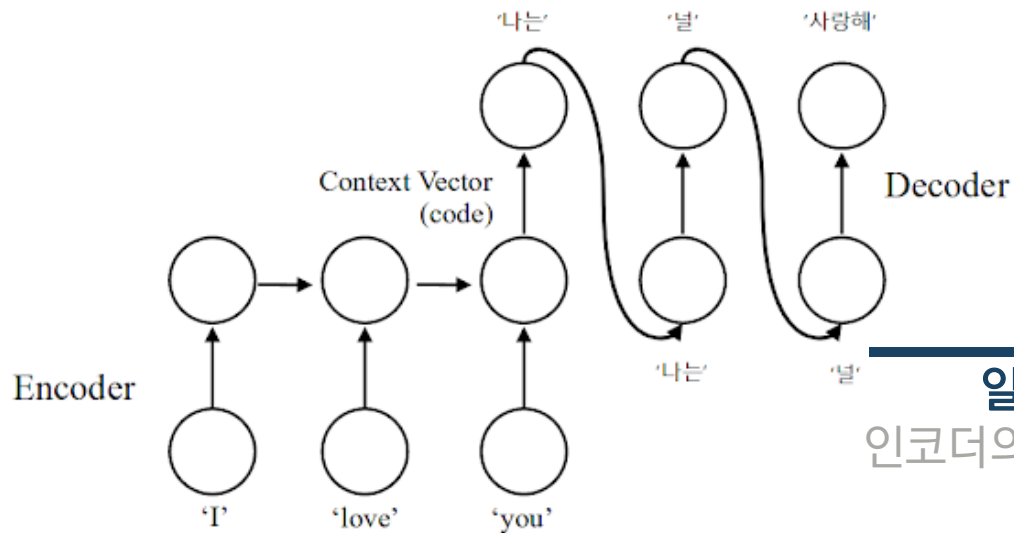


중요한 부분에 집중하는 법은 없을까 ?



Attention Mechanism으로 해결 !



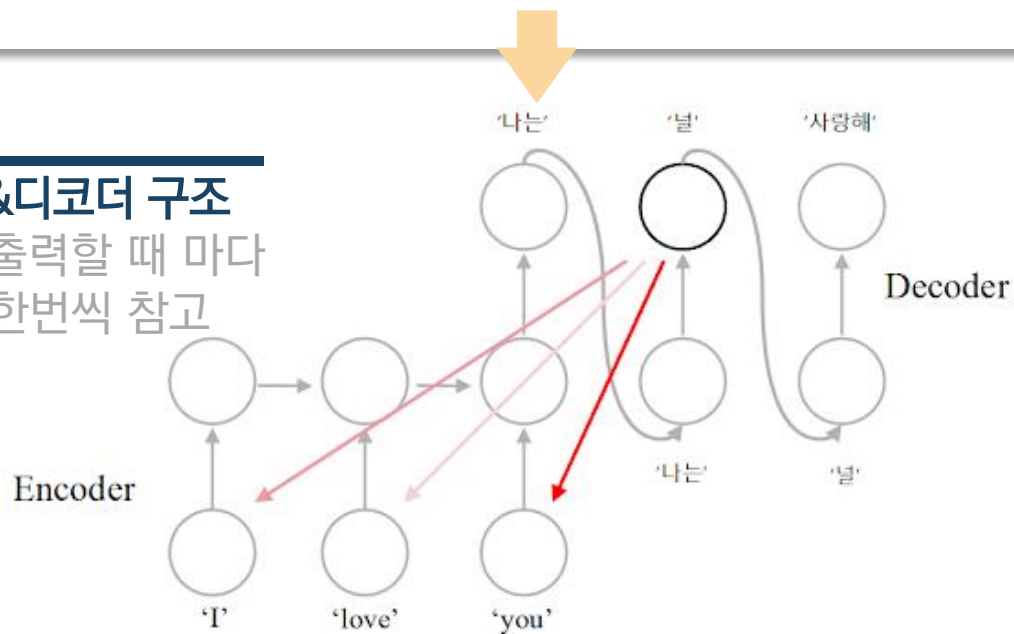


**일반 인코더&디코더 구조**  
 인코더의 마지막(hidden state)이  
 그대로 디코더로 입력



## 어텐션 적용 인코더&디코더 구조

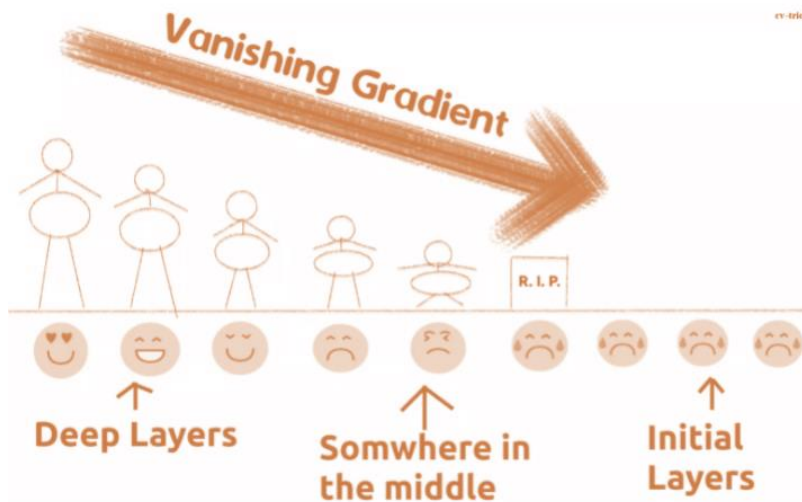
디코더에서 단어를 출력할 때 마다  
 입력 sequence를 한번씩 참고



# About Attention

## 정보 손실

하나의 고정된 크기 벡터에  
모든 정보를 압축할 수 없어



기울기 소실(vanishing gradient) 문제  
순차적 데이터 학습의  
고질적인 문제

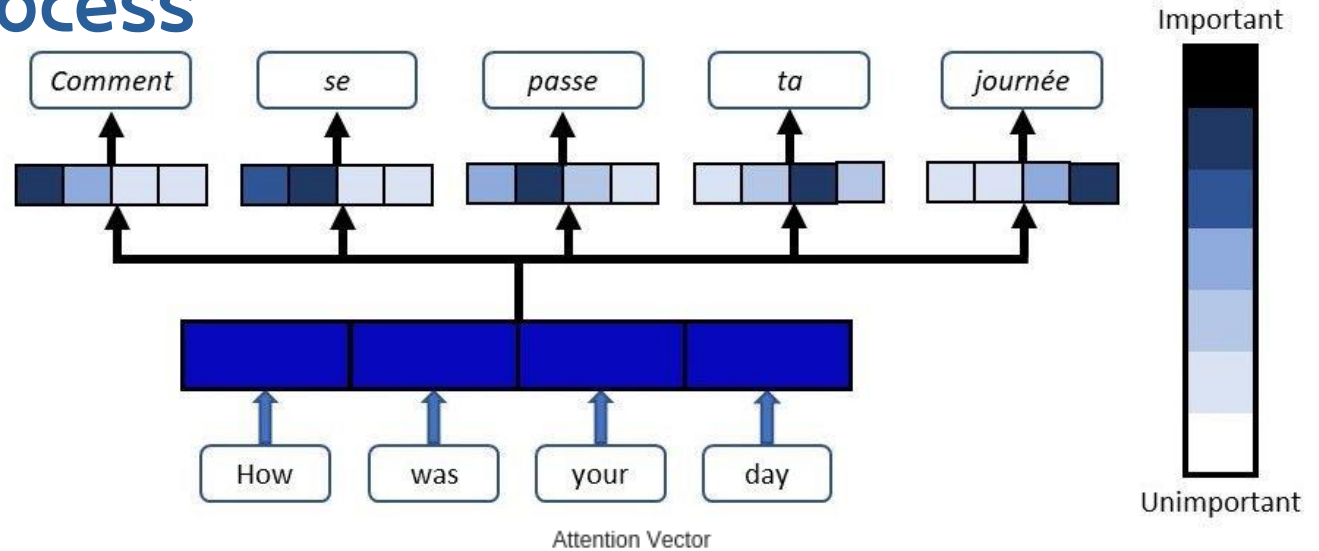
## 정확도 ▲

Seq2seq 등의 기계번역에서의  
정확도 향상

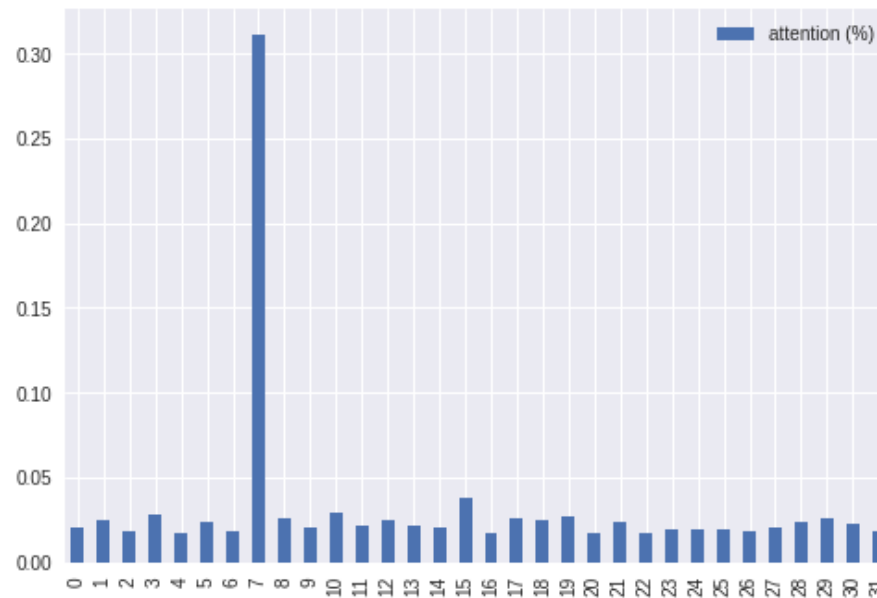


Concentrate to  
Important things !

# Attention process



With matplotlib

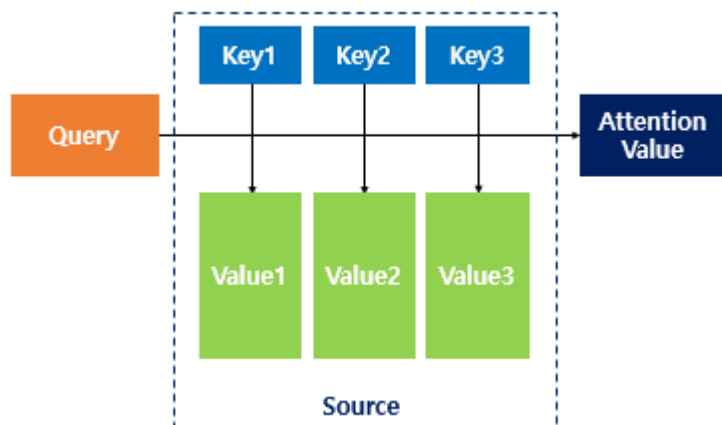
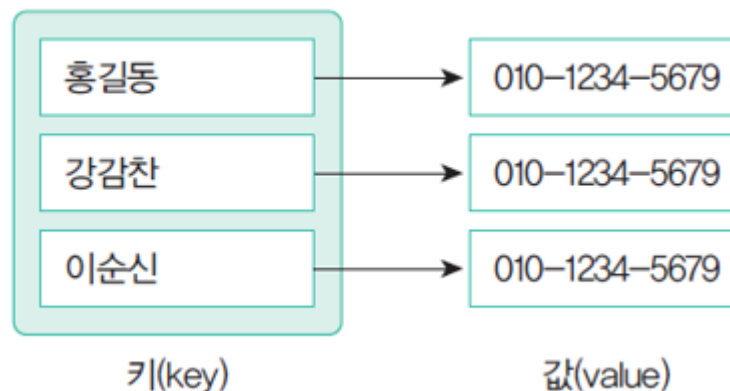




# About Attention's mechanism & function

일반적인 dictionary 구조 ▶

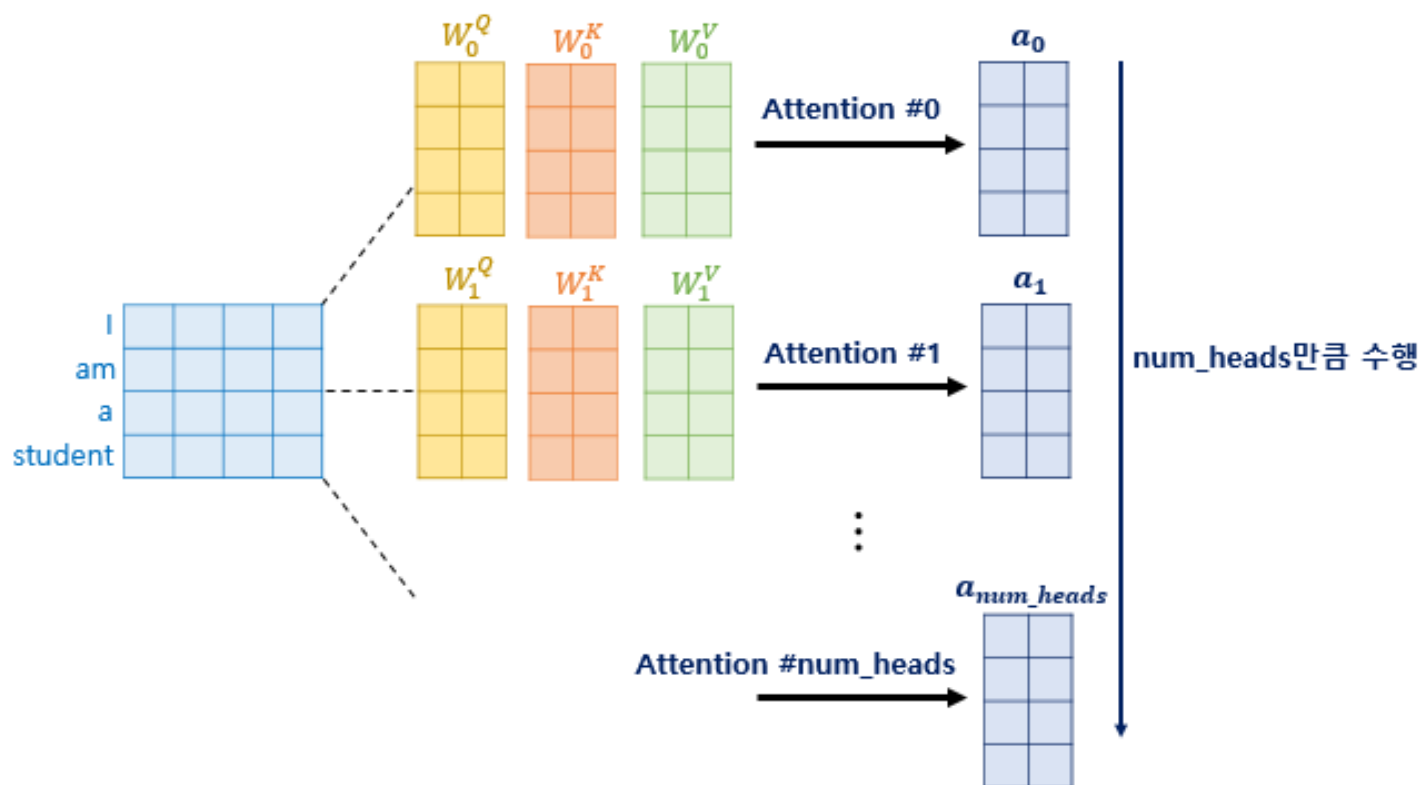
딕셔너리



**"Basic function of Attention"**

▶  $\text{Attention}(Q, K, V) = \text{Attention Value}$

# If it is a Multihead-Attention architecture



# If the model is a Seq2seq ?

→ You can set Attentions  
by adjusting the functions

Name	Alignment score function	Citation
Content-base attention	$\text{score}(s_t, h_i) = \text{cosine}[s_t, h_i]$	<a href="#">Graves2014</a>
Additive(*)	$\text{score}(s_t, h_i) = \mathbf{v}_a^\top \tanh(\mathbf{W}_a[s_t; h_i])$	<a href="#">Bahdanau2015</a>
Location-Base	$\alpha_{t,i} = \text{softmax}(\mathbf{W}_a s_t)$ Note: This simplifies the softmax alignment to only depend on the target position.	<a href="#">Luong2015</a>
General	$\text{score}(s_t, h_i) = s_t^\top \mathbf{W}_a h_i$ where $\mathbf{W}_a$ is a trainable weight matrix in the attention layer.	<a href="#">Luong2015</a>
Dot-Product	$\text{score}(s_t, h_i) = s_t^\top h_i$	<a href="#">Luong2015</a>
Scaled Dot-Product(^)	$\text{score}(s_t, h_i) = \frac{s_t^\top h_i}{\sqrt{n}}$ Note: very similar to the dot-product attention except for a scaling factor; where n is the dimension of the source hidden state.	<a href="#">Vaswani2017</a>

# In Attention is all you need thesis

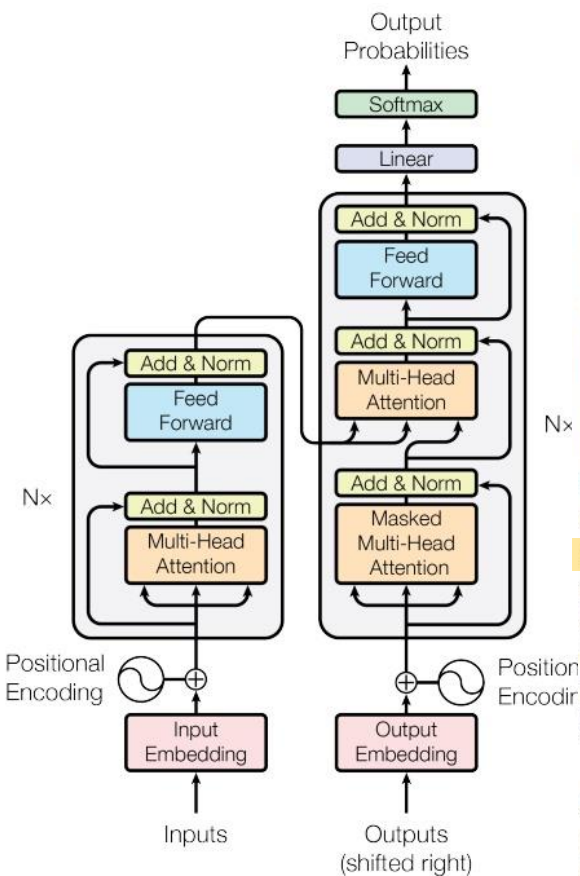
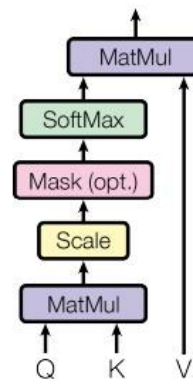
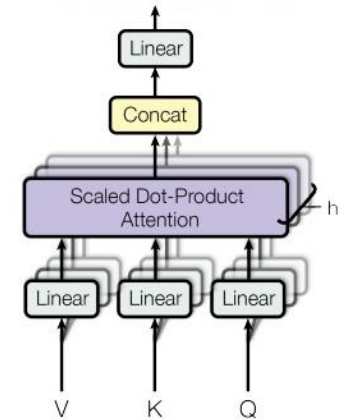


Figure 1: The Transformer - model architecture.

## Scaled Dot-Product Attention



## Multi-Head Attention



### 3.1 Encoder and Decoder Stacks

**Encoder:** The encoder is composed of a stack of  $N = 6$  identical layers. Each layer has two sub-layers. The first is a **multi-head self-attention** mechanism, and the second is a simple, position-wise fully connected feed-forward network. We employ a residual connection [11] around each of the two sub-layers, followed by layer normalization [1]. That is, the output of each sub-layer is  $\text{LayerNorm}(x + \text{Sublayer}(x))$ , where  $\text{Sublayer}(x)$  is the function implemented by the sub-layer itself. To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce outputs of dimension  $d_{\text{model}} = 512$ .

**Decoder:** The decoder is also composed of a stack of  $N = 6$  identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs **multi-head attention** over the output of the encoder stack. Similar to the encoder, we employ residual connections around each of the sub-layers, followed by layer normalization. We also modify the **self-attention** sub-layer in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with fact that the output embeddings are offset by one position, ensures that the predictions for position  $i$  can depend only on the known outputs at positions less than  $i$ .

### 3.2 Attention

An attention function can be described as mapping a query and a set of key-value pairs to an output, where the query, keys, values, and output are all vectors. The output is computed as a weighted sum of the values, where the weight assigned to each value is computed by a compatibility function of the query with the corresponding key.

How many attentions does bert have?

전체

뉴스

이미지

동영상

쇼핑

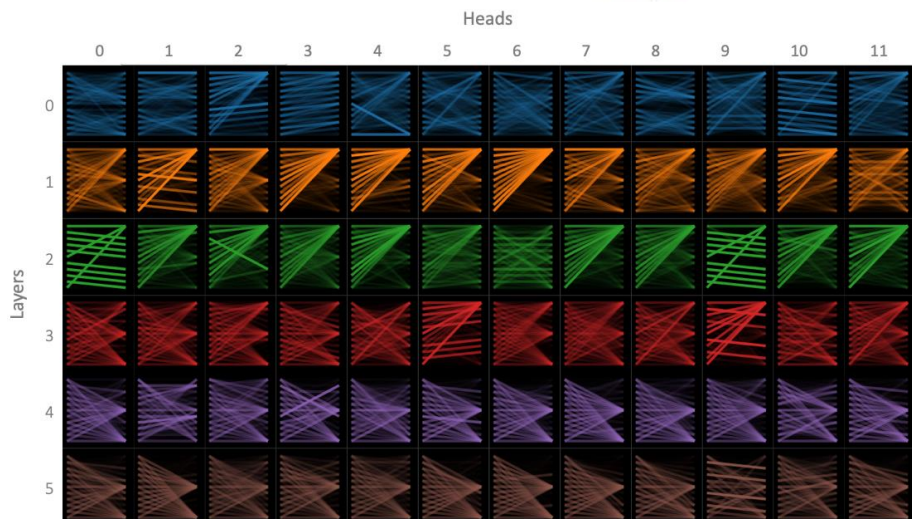
더보기

도구

검색결과 약 19,300,000개 (0.55초)

도움말: **한국어** 검색결과만 검색합니다. 환경설정에서 검색 언어를 지정할 수 있습니다.

The version of BERT that we consider here — BERT Base — has 12 layers and 12 heads, resulting in a total of  $12 \times 12 = 144$  distinct attention mechanisms. We can visualize attention in all of the heads at once, using the model view (available in interactive form here):



deconstructing-bert-part-...

RT, Part 2: Visualizing the Inner Workings ...



☐ Chapter: 1

☒ Chapter: 2

☐ Chapter: 3

☐ Chapter: 4

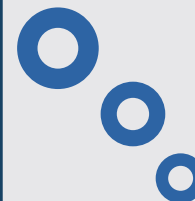
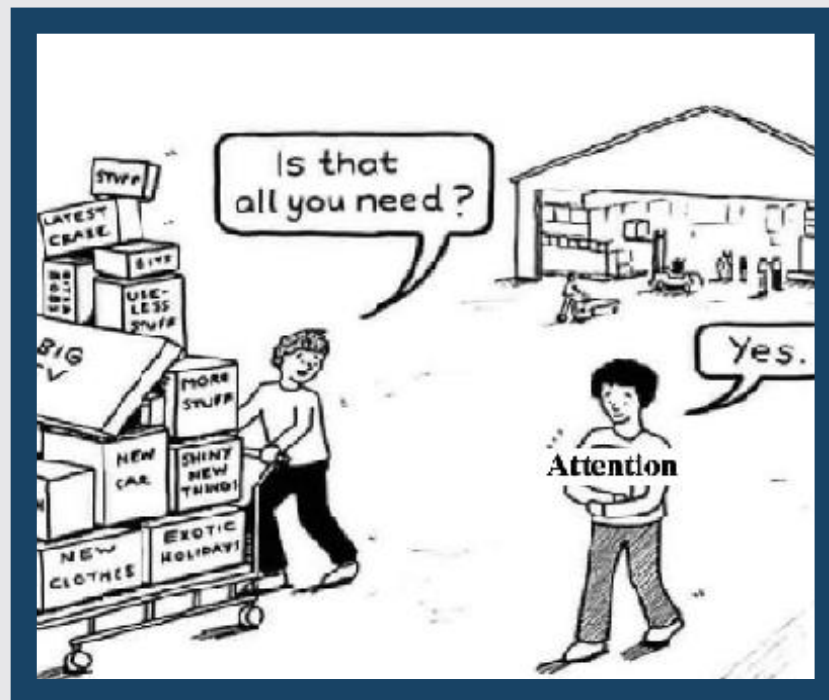
Chapter

# Transformer&code

알고리즘 및 코드 해석







## Attention Is All You Need

Ashish Vaswani\*  
Google Brain  
avaswani@google.com

Noam Shazeer\*  
Google Brain  
noam@google.com

Niki Parmar\*  
Google Research  
nikip@google.com

Jakob Uszkoreit\*  
Google Research  
usz@google.com

Llion Jones\*  
Google Research  
llion@google.com

Aidan N. Gomez\*<sup>†</sup>  
University of Toronto  
aidan@cs.toronto.edu

Lukasz Kaiser\*  
Google Brain  
lukaszkaizer@google.com

Illia Polosukhin\*<sup>‡</sup>  
illia.polosukhin@gmail.com

### Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.



# Transformer

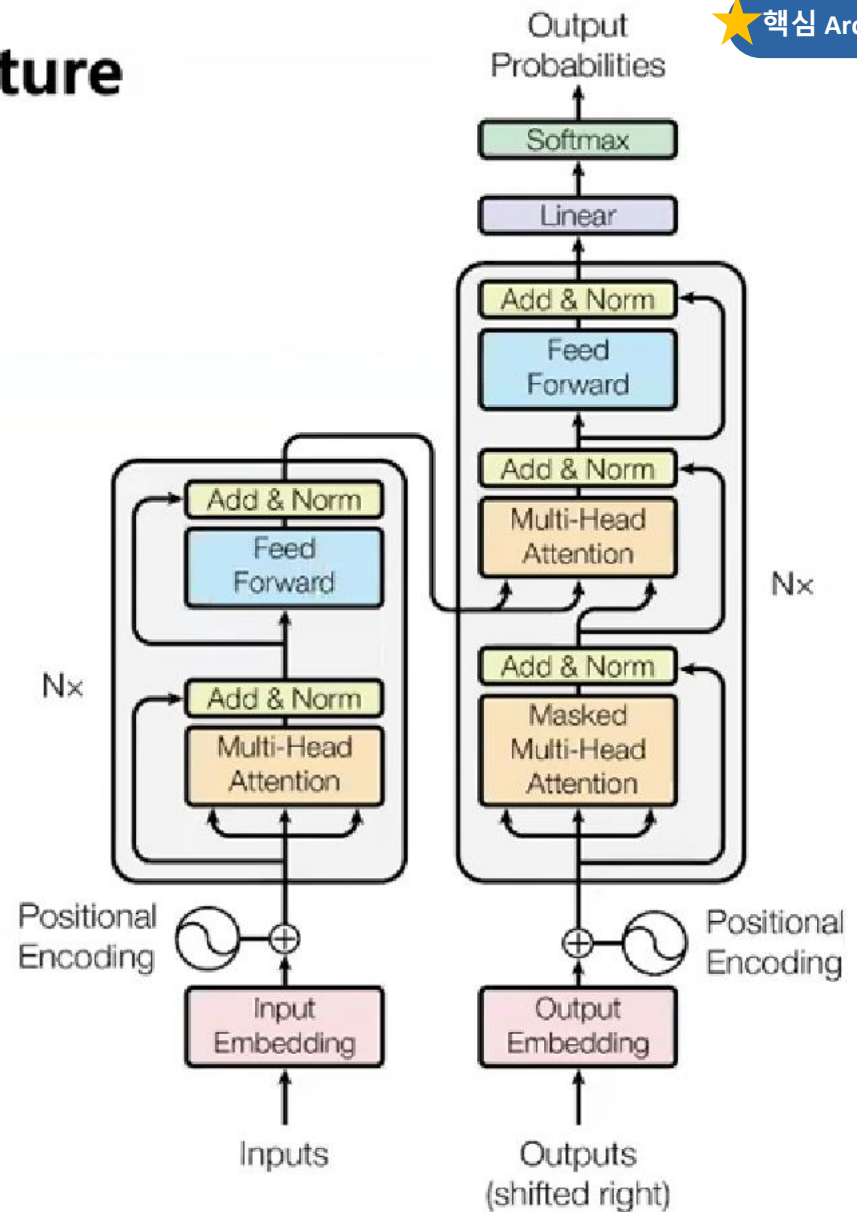
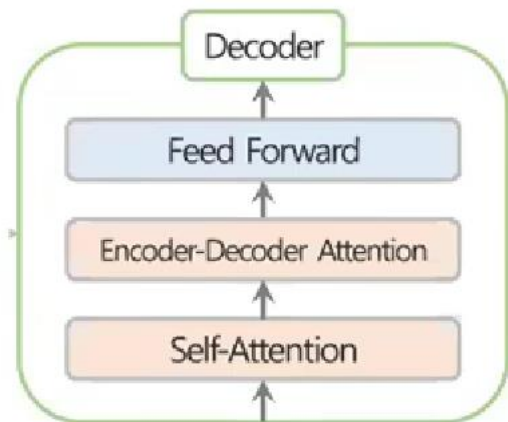
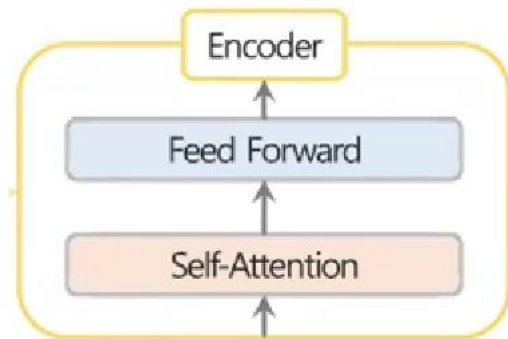
“ Attention ”  
is all you need

from GOOGLE TEAM

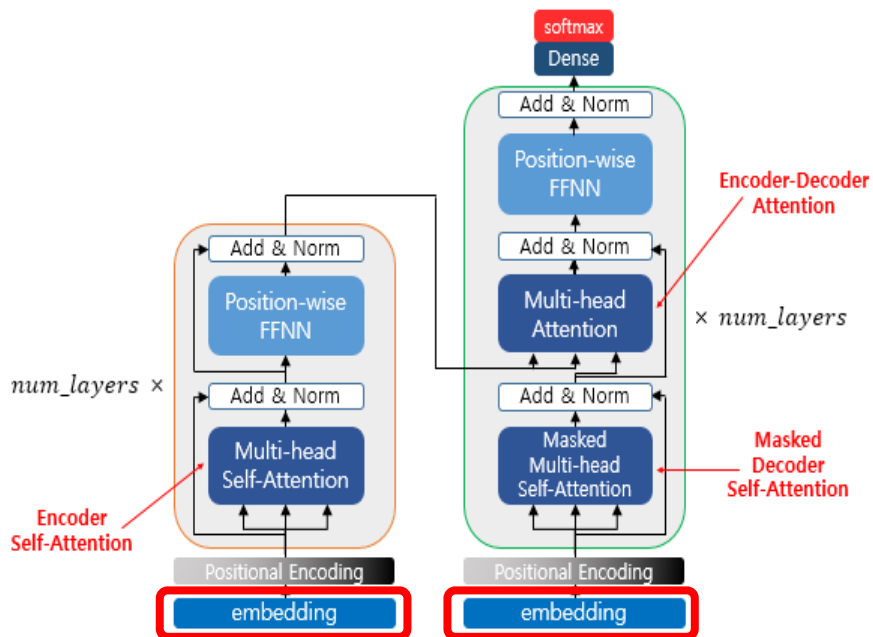




# Transformer architecture



## # 알고리즘 생성부



Index	Word	Ont-hot vector	Embedding vector
0	영웅은	[1 0 0 0 0]	[0.1 4.2 1.5 1.1 2.8]
1	죽지	[0 1 0 0 0]	[1.0 3.1 2.5 0.7 1.1]
2	않아요	[0 0 1 0 0]	[0.3 2.1 1.5 2.1 0.1]
3	너만	[0 0 0 1 0]	[2.2 1.4 0.5 0.9 1.1]
4	빼고	[0 0 0 0 1]	[0.7 1.7 0.5 0.3 0.2]

## • 임베딩 - 서브워드토큰라이저

간단하게 encode decode하고 싶지만  
OOV문제 발생 (Out-Of-Vocabulary)

*Can use these cases*

- SentencePiece
- Tokenizer(subword-text-encoder)

# Datasets & Preprocessing

## to Sentencepiece

### 위키백과:데이터베이스 다운로드

위키백과, 우리 모두의 백과사전.



이 문서는 한국어 위키백과의 정보문입니다.

이 문서는 정책과 지침은 아니지만, 위키백과의 규범과 관습 측면에서 공동체의 확립된 관행을 설명하고 있습니다. 공동체의 총의나 검토를 어느 정도 반영하고 있습니다.

단축  
백:다운로드



이 문서는 데이터베이스 다운로드에 관한 것입니다. 데이터베이스 보고서에 대해서는 위키백과:데이터베이스 보고서 문서를 참고하십시오.

위키백과의 자료를 여러가지 용도로 이용하려는 사람들을 위해, 위키백과에서는 주기적으로 전체 문서를 묶어서 배포하고 있습니다.

여기에서 한국어 위키백과 덤프를 받을 수 있습니다.



> 외부 데이터 > KETIR&D데이터 > 위키백과 데이터 > 위키백과 데이터

## To Tokenizer

### 웰니스 대화 스크립트 데이터셋



● 외부 데이터는 해당 기관의 이용정책과 다운로드 절차를 따라야 하며 AI 허브와 관련이 없음을 알려 드립니다. [저작권 및 이용정책 상세보기]

#### 구축량

- 정신건강 상담 주제의 359개 대화의도에 대한 5,232개의 사용자 발화 및 1,023개의 챗봇 발화 구축

#### 대표 도면

대분류	소분류	사용자 발화	챗봇 발화
감정	감정조절 이상	더 이상 내 감정을 내가 컨트롤 못하겠어.	감정이 조절이 안 될 때만큼 힘들 때는 없는 거 같아요.
감정	걱정	계속 이렇까 봐 너무 걱정돼.	모든 문제는 해결되기 마련이잖아요. 마음을 편히 드세요.
감정	자괴감	저는 왜 이렇게 못한 인간일까요...	조금 돌아가는 것뿐이라고 생각해요. 기운내세요..
배경	남편	남편이랑 한달에 몇번은 보는 것 같아요.	남편 분과 갈등을 겪고 계시군요. 마음이 많이 상하셨겠어요.
배경	대인관계	중학교 친구랑 가족 말고는 연락하는 사람도 없어.	사람 대하는 게 가장 어려운 문제인 거 같아요. 힘들죠?

# Sentencepiece (실습)

```
# 구글-센텐스피스
import sentencepiece as spm
import pandas as pd

import sys
sys.stdout = open('wiki_sample.txt', 'w', encoding='utf8') #stdout으로 출력물을 감지
print(efb.text)
sys.stdout.close() #close하며 출력물을 저장
```

## 사용예제

```
spm.SentencePieceTrainer.train(
    f"--input={corpus} --model_prefix={prefix} --vocab_size={vocab_size + 7}" +
    " --model_type=bpe" +
    " --max_sentence_length=9999")

# " --max_sentence_length=9999" + # 문장 최대 길이
# " --pad_id=0 --pad_piece=[PAD]" + # pad (0)
# " --unk_id=1 --unk_piece=[UNK]" + # unknown (1)
# " --bos_id=2 --bos_piece=[BOS]" + # begin of sequence (2)
# " --eos_id=3 --eos_piece=[EOS]" + # end of sequence (3)
# " --user_defined_symbols=[SEP],[CLS],[MASK]" # 사용자 정의 토큰

corpus = "wiki_sample.txt"
prefix = "wiki_sample"
vocab_size = 500
spm.SentencePieceTrainer.Train('--input=wiki_sample.txt --model_prefix=wiki_sample --vocab_size=500 --model_type=bpe --max_sentence_length=9999')
```

BeautifulSoup



Data (18.1Gb)

Data (2Kb)

```
# vocab 생성이 완료되면 imdb.model, imdb.vocab 파일 두개가 생성됨.
# vocab 파일에서 학습된 서브워드들을 확인할 수 있음.\
# 단어 집합의 크기를 확인하기 위해 vocab 파일을 데이터프레임에 저장함.
vocab_list = pd.read_csv('wiki_sample.vocab', sep='\t', header=None)
vocab_list.sample(10)
print(len(vocab_list))
```

```
# model 파일을 로드
sp = spm.SentencePieceProcessor()
vocab_file = "wiki_sample.model"
sp.load(vocab_file) # True
```

# Sentencepiece (실습)

```
'-' 'd:\personal_project\make_vocab.py'
```

John D. Hunter (August 1, 1968 – August 28, 2012) was an American neurobiologist and the original author of Matplotlib. He studied initially at Princeton University, later he obtained a Ph.D. in neurobiology from the University of Chicago in 2000. He was a founding director of NumFOCUS Foundation. Matplotlib was originally conceived to visualize electrocorticography data. The software tool emerged as the most widely used plotting library for the Python programming language, and a core component of the data visualization during landing of the Phoenix spacecraft in 2008 as well as for the creation of the first image of a black hole. He died on August 28, 2012. His memorial service was held at the University of Chicago's Rockefeller Chapel (which was his alma mater), and three daughters: Clara, Ava, and Rahel. A memorial fund to honor his work has been established to help with the needs of the Foundation. The Distinguished Service Award – the Foundation's highest honor – to Hunter in order to recognize his long-term excellence in the Python community. He won the Excellence in Plotting Contest in his honor to continue the advancement of scientific plotting, where the first prize winner was John D. Hunter.

```
# 문장 확인용
```

```
lines = [
    "the Foundation's highest honor",
]
for line in lines:
    print(line)
    print(sp.encode_as_pieces(line))
    print(sp.encode_as_ids(line))
    print()
```

```
# 출력물
```

```
the Foundation's highest honor
['_the', '_Foundation', "'", 's', '_hig', 'hest', '_honor']
[13, 145, 480, 443, 409, 379, 140]
```

```
PS D:\personal_project> █
```

```
# vocab loading
```



# Tokenizer (본문)

(subword-text-encoder)

```
answers = []
for sentence in train_data['A']:
    # 구두점에 대해서 띄어쓰기
    # ex) 12시 땡! -> 12시 땡 !
    sentence = re.sub(r"([?.!,])", r" \1 ", sentence)
    sentence = sentence.strip()
    answers.append(sentence)
questions = []
for sentence in train_data['Q']:
    sentence = re.sub(r"([?.!,])", r" \1 ", sentence)
    sentence = sentence.strip()
    questions.append(sentence) # sentence를 array 맨 끝에 추가

tokenizer = tfds.deprecated.text.SubwordTextEncoder.build_from_corpus(
    questions + answers, target_vocab_size=2**13)
```

└ 임의로 넣어봄

└ 토큰화할 데이터를 넣어줌

```
# 임의로 선택한 20번 질문 샘플. 즉, questions[20]을 가지고 정수 인코딩을 진행해본다
print('Tokenized sample question: {}'.format(tokenizer.encode(questions[20])))
# 임의의 질문 샘플을 정수 인코딩 : [5766, 611, 3509, 141, 685, 3747, 849]

# 서브워드텍스트인코더 토큰라이저의 .encode()와 decode() 테스트해보기

# 임의의 입력 문장을 sample_string에 저장
sample_string = questions[20]

# encode() : 텍스트 시퀀스 --> 정수 시퀀스
tokenized_string = tokenizer.encode(sample_string)
print('정수 인코딩 후의 문장 {}'.format(tokenized_string))

# decode() : 정수 시퀀스 --> 텍스트 시퀀스
original_string = tokenizer.decode(tokenized_string)
print('기존 문장: {}'.format(original_string))
# 정수 인코딩 후의 문장 [5766, 611, 3509, 141, 685, 3747, 849]
# 기존 문장: 가스비 비싼데 감기 걸리겠어
```



Data (1Mb)

웹니스 대화 스크립트 데이터셋

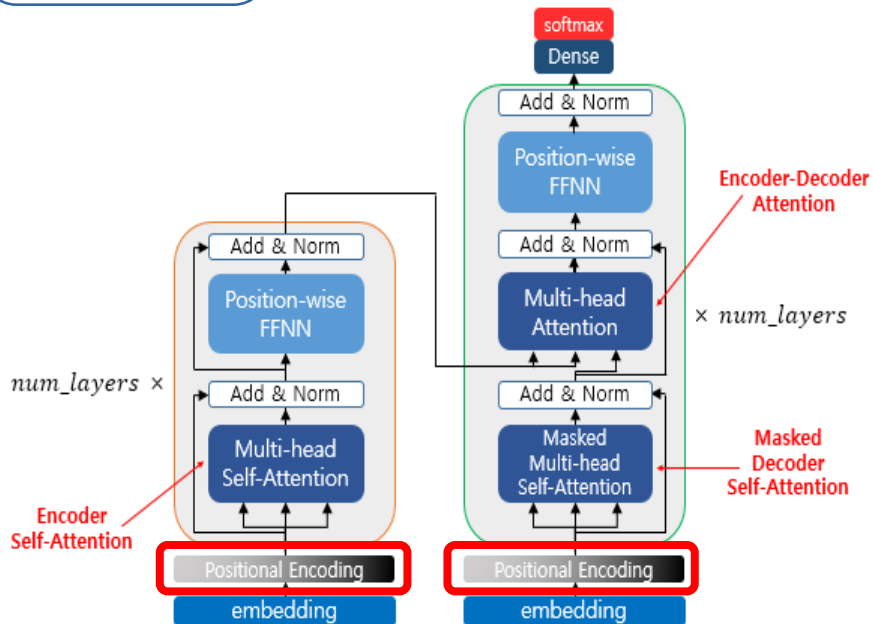
# Tokenizer (본문)

(subword-text-encoder)

```
# 패딩
tokenized_inputs = tf.keras.preprocessing.sequence.pad_sequences(
    tokenized_inputs, maxlen=MAX_LENGTH, padding='post')
# 정수 인코딩과 패딩이 진행된 후의 데이터의 크기를 확인한다.
print('질문 데이터의 크기(shape) :', questions.shape)
# 질문 데이터의 크기(shape) : (11823, 40)
print('답변 데이터의 크기(shape) :', answers.shape)
# 답변 데이터의 크기(shape) : (11823, 40)

# 0번째 샘플을 임의로 출력(길이 40을 맞추기 위해 뒤에 0이 패딩된 것을 확인할 수 있다.)
print(questions[0])
# [8178 7915 4207 3060 41 8179 0 0 0 0 0 0 0 0]
# 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0 0 0 0]
print(answers[0])
# [8178 3844 74 7894 1 8179 0 0 0 0 0 0 0 0]
# 0 0 0 0 0 0 0 0 0 0 0 0 0 0
# 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

↳패딩해줌



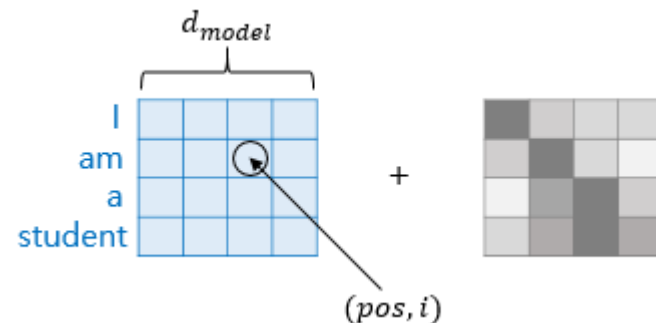
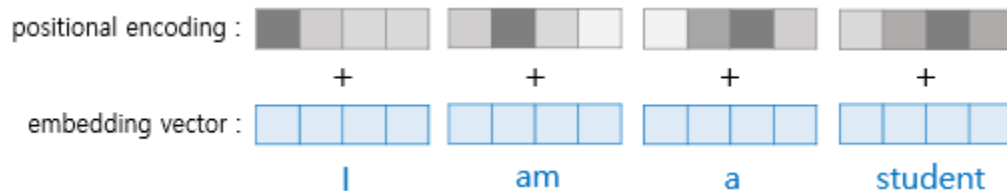
## • 포지셔널 인코딩

. 각 단어의 위치정보 보존

. 공식

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

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





# Positional Encoding

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

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

```
class PositionalEncoding(layers.Layer):
    def __init__(self, position, d_model):
        super(PositionalEncoding, self).__init__()
        self.pos_encoding = self.positional_encoding(position, d_model)

    def get_angles(self, position, i, d_model):
        angles = 1 / tf.pow(10000, (2 * (i // 2)) / tf.cast(d_model, tf.float32))
        return position * angles

    def positional_encoding(self, position, d_model):
        angle_rads = self.get_angles(
            position=tf.range(position, dtype=tf.float32)[:], tf.newaxis],
            i=tf.range(d_model, dtype=tf.float32)[tf.newaxis, :],
            d_model=d_model)
```

```
# 배열의 홀수 인덱스(2i+1)에는 코사인 함수 적용
cosines = tf.math.cos(angle_rads[:, 1::2])
```

```
angle_rads = np.zeros(angle_rads.shape)
angle_rads[:, 0::2] = sines
angle_rads[:, 1::2] = cosines
pos_encoding = tf.constant(angle_rads)
pos_encoding = pos_encoding[tf.newaxis, ...]
```

```
print(pos_encoding.shape)
return tf.cast(pos_encoding, tf.float32)
```

```
def call(self, inputs):
    return inputs + self.pos_encoding[:, :tf.shape(inputs)[1], :]
```

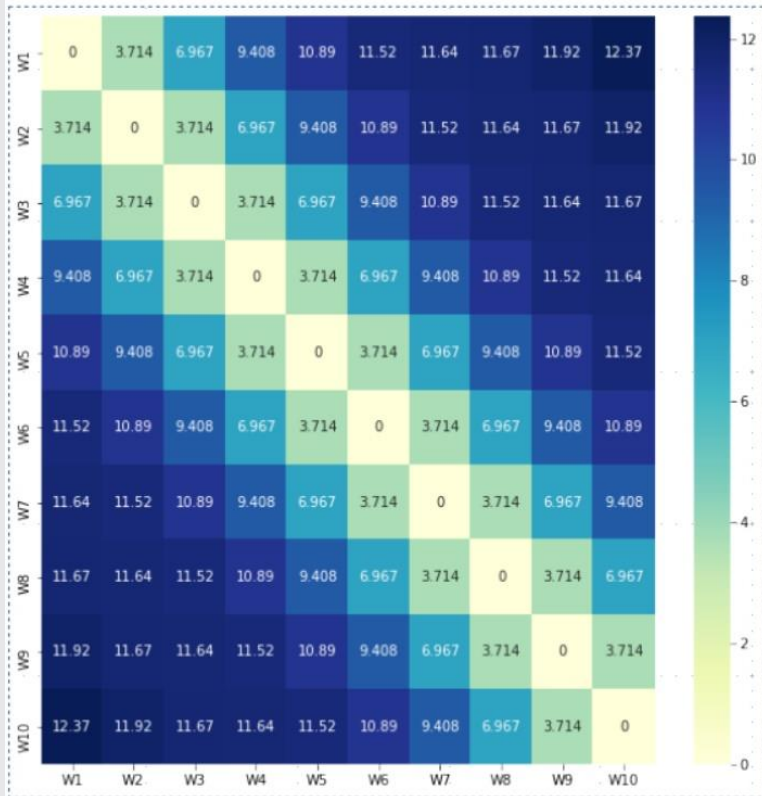
```
# 50 × 128의 크기를 가지는 포지셔널 인코딩 행렬을 시각화하여 어떤 형태를 가지는지 확인
sample_pos_encoding = PositionalEncoding(50, 128)
```

```
plt.pcolormesh(sample_pos_encoding.pos_encoding.numpy()[0], cmap='RdBu')
plt.xlabel('Depth')
plt.xlim((0, 128))
plt.ylabel('Position')
plt.colorbar()
plt.show()
```

← 시각화하여  
포지셔널 인코딩 확인

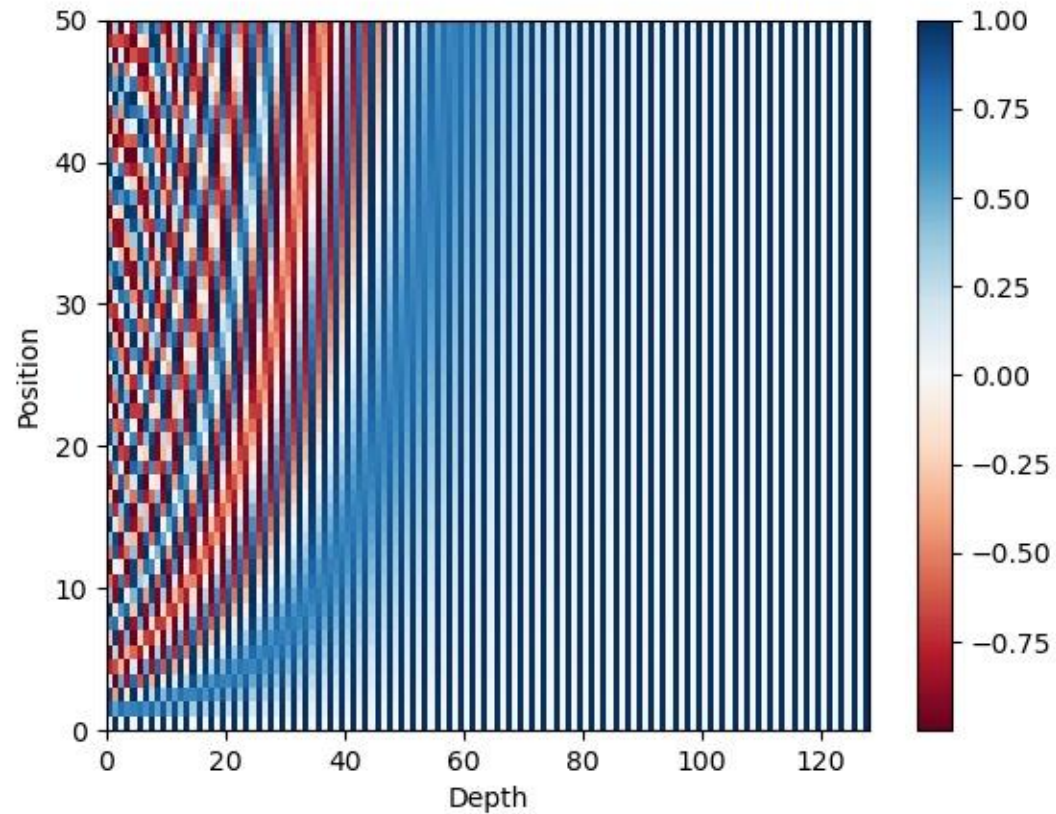
# Positional Encoding

10개의 단어로 이루어진 문장



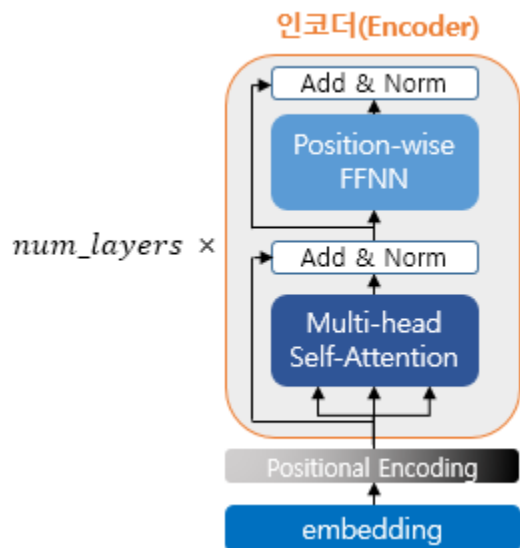
$10 * 10$

$50 * 128$

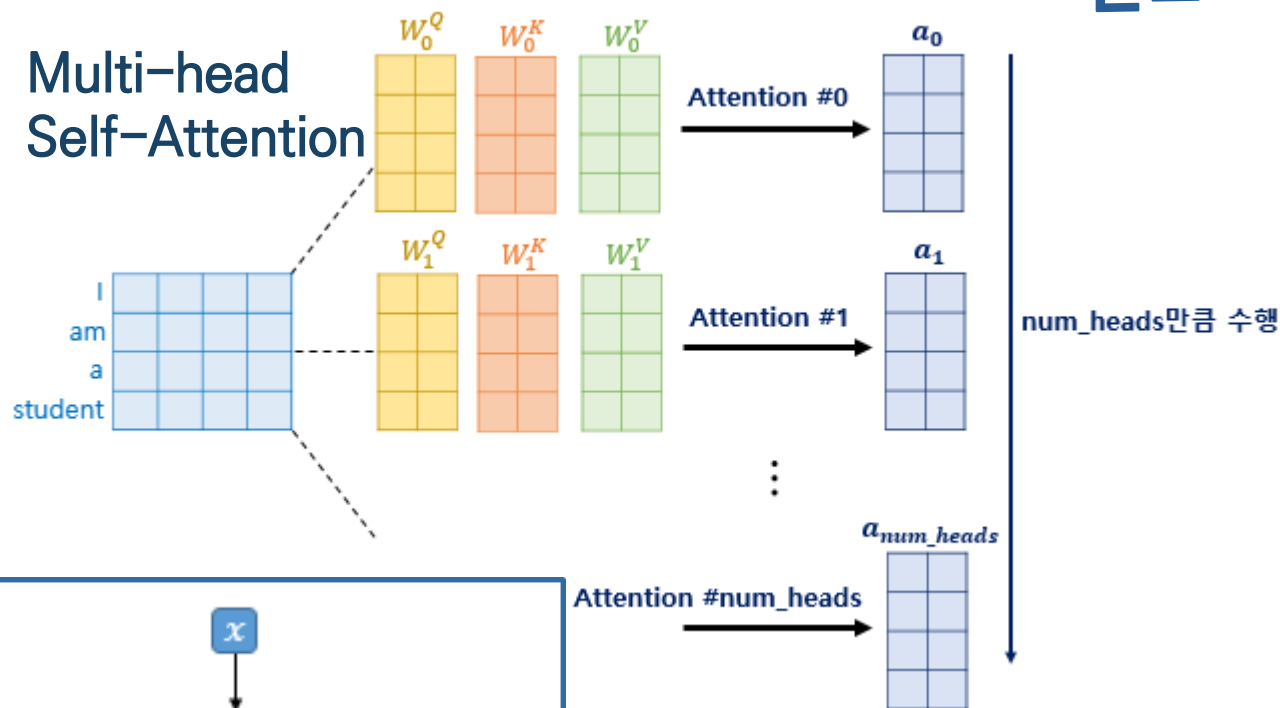


## # 알고리즘 생성부

## · 인코더

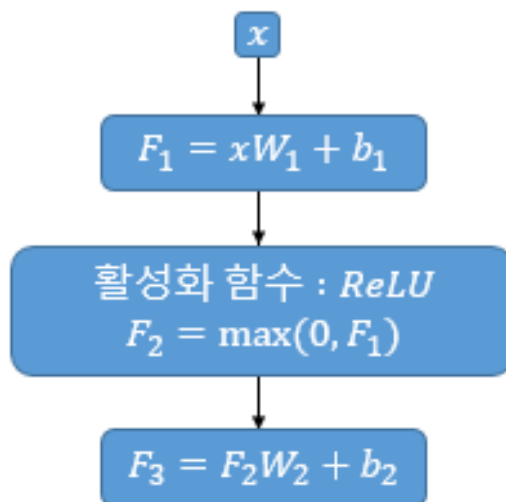


## Multi-head Self-Attention



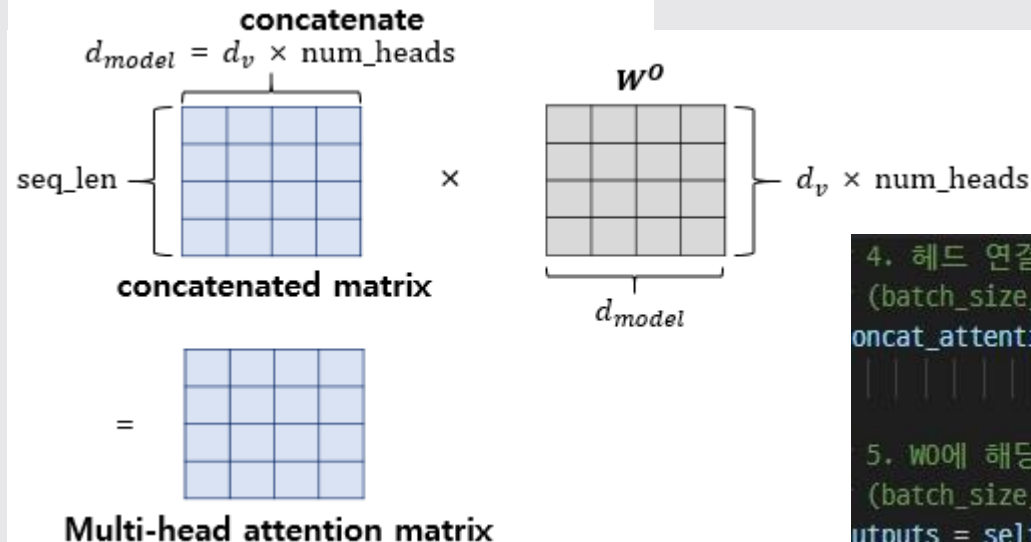
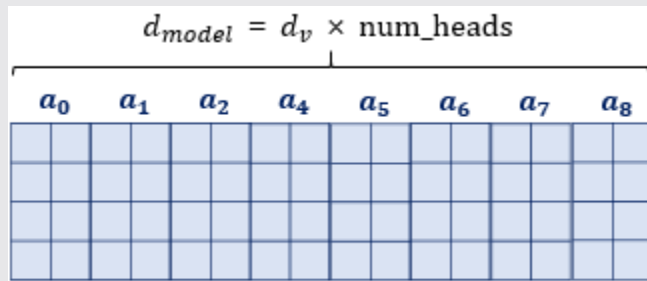
The animal didn't cross the street because it was too tired.

The animal didn't cross the street because it was too tired.



## Position-wise FFNN

# Multihead-Attention



```
# d_model을 num_heads로 나눈 값.
# 논문 기준 : 64
self.depth = d_model // self.num_heads

# WQ, WK, WV에 해당하는 밀집층 정의
self.query_dense = tf.keras.layers.Dense(units=d_model)
self.key_dense = tf.keras.layers.Dense(units=d_model)
self.value_dense = tf.keras.layers.Dense(units=d_model)

# W0에 해당하는 밀집층 정의
self.dense = tf.keras.layers.Dense(units=d_model)
```

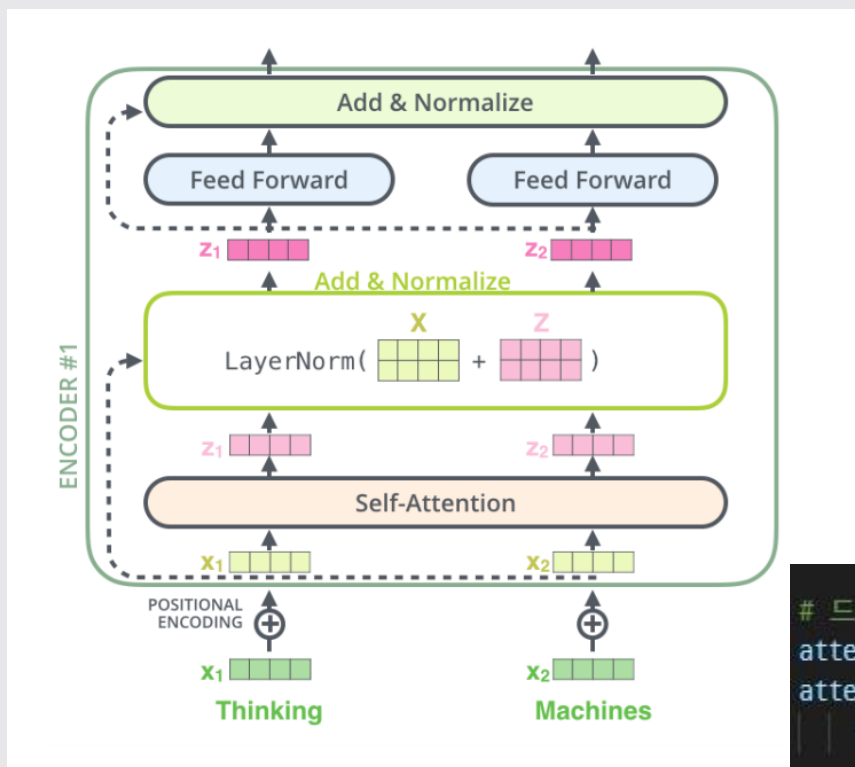
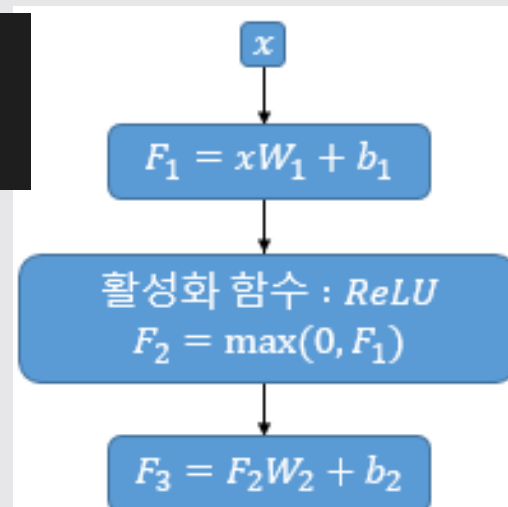
4. 헤드 연결(concatenate)하기  
(batch\_size, query의 문장 길이, d\_model)  
concat\_attention = tf.reshape(scaled\_attention,  
| | | | | | | | | | (batch\_size, -1, self.d\_model))

5. W0에 해당하는 밀집층 지나기  
(batch\_size, query의 문장 길이, d\_model)  
outputs = self.dense(concat\_attention)

$$\text{softmax} \left( \frac{Q \times K^T}{\sqrt{d_k}} \right) \times V = \text{Attention Value Matrix } a$$

# Position-wise FFNN

```
# 포지션 와이즈 피드 포워드 신경망 (두번째 서브층)
outputs = tf.keras.layers.Dense(units=dff, activation='relu')(attention)
outputs = tf.keras.layers.Dense(units=d_model)(outputs)
```

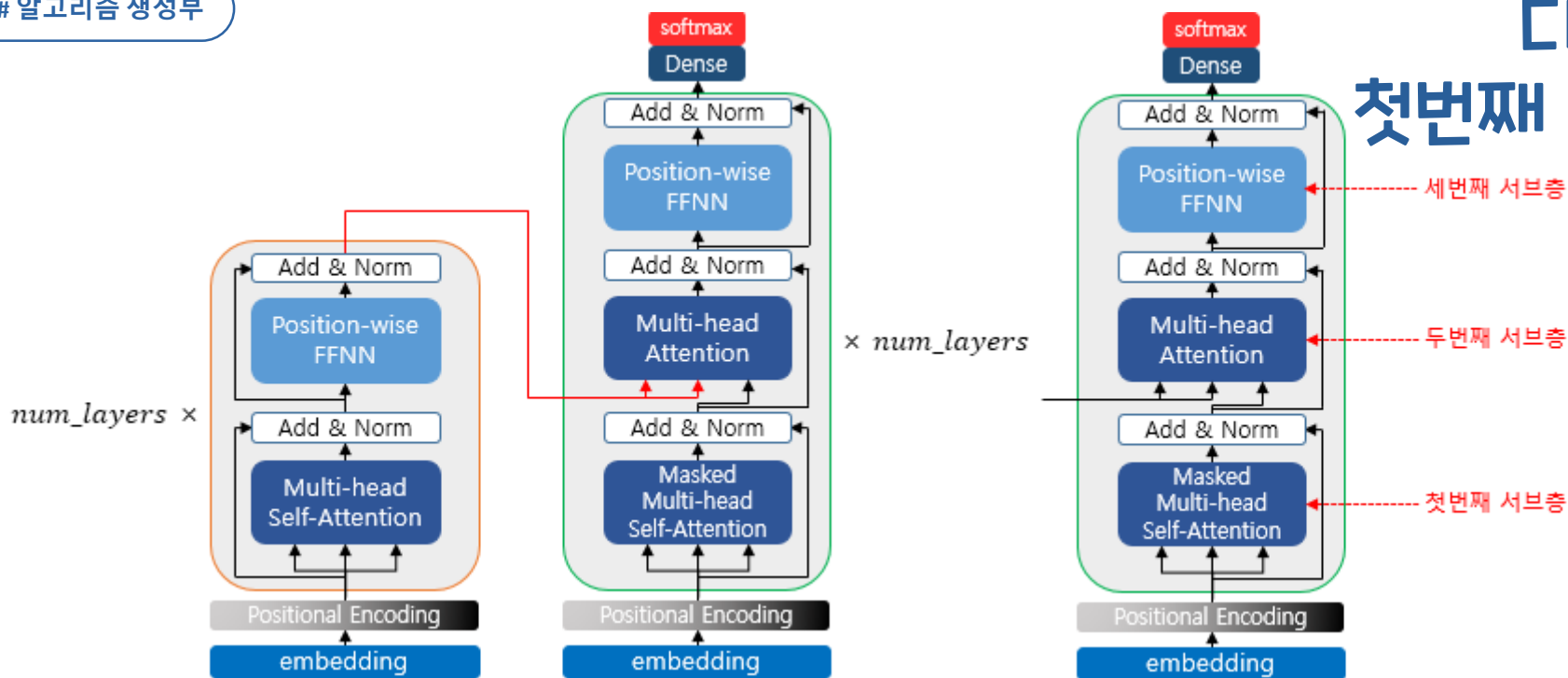


## Add&Norm

```
# 드롭아웃 + 잔차 연결과 층 정규화
attention = tf.keras.layers.Dropout(rate=dropout)(attention)
attention = layers.LayerNormalization(
    epsilon=1e-6)(inputs + attention)
```



## # 알고리즘 생성부

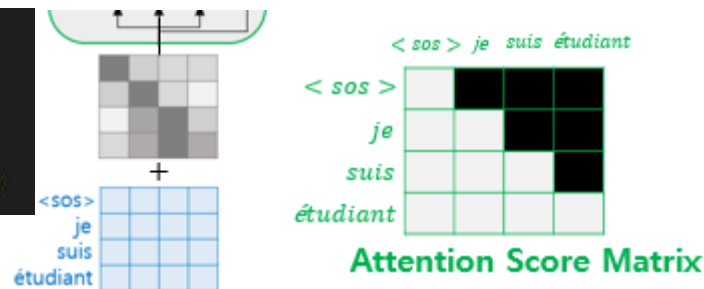


## 디코더-첫번째 서브층

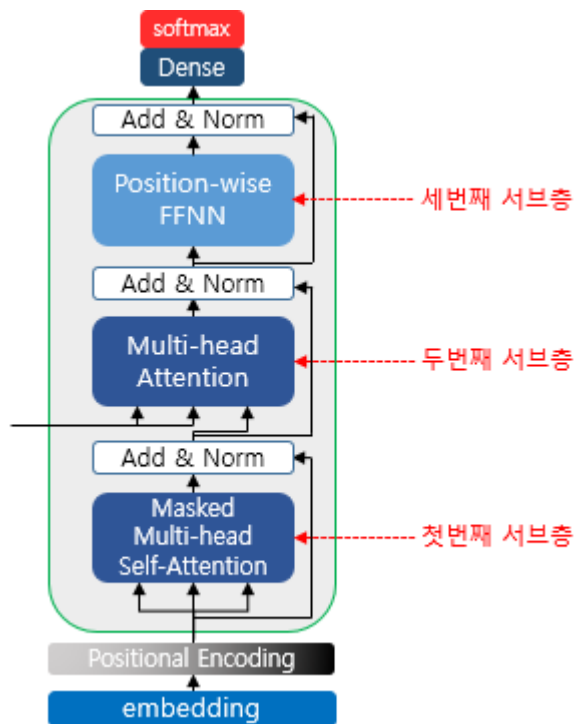
##### 디코더 구현부

```
def decoder_layer(dff, d_model, num_heads, dropout, name="decoder_layer"):
    inputs = tf.keras.Input(shape=(None, d_model), name="inputs")
    enc_outputs = tf.keras.Input(shape=(None, d_model), name="encoder_outputs")

    look_ahead_mask = tf.keras.Input(
        shape=(1, None, None), name="look_ahead_mask")
    padding_mask = tf.keras.Input(shape=(1, 1, None), name='padding_mask')
```



## 디코더- 두,세번째 서브층



```
# 멀티-헤드 어텐션 (첫번째 서브층 / 마스크드 셀프 어텐션)
attention1 = MultiHeadAttention(
    d_model, num_heads, name="attention_1")(inputs={
        'query': inputs, 'key': inputs, 'value': inputs, # Q = K = V
        'mask': look_ahead_mask # 룩어헤드 마스크
    })
```

```
# 드롭아웃 + 잔차 연결과 층 정규화
attention = tf.keras.layers.Dropout(rate=dropout)(attention)
attention = layers.LayerNormalization(
    epsilon=1e-6)(inputs + attention)
```

☐ Chapter: 1

☐ Chapter: 2

☒ Chapter: 3

☐ Chapter: 4

# Result

결과물 - 챗봇 구현부

Chapter





# 전처리 데이터 in DataBase

```
# 데이터 전부를 sql로 import한다 (SQL상 기본 컬럼 및 세팅은 이미 SQL-WORKBENCH 상에서 완료한 상태, 사실 이 작업도 워크bench에서 하는게 편함)
import_data = "LOAD DATA LOCAL INFILE 'd:/personal_project/_data/ChatBotData_ANSI.csv' +
"INTO TABLE transformer_chatbot.chatbot_table FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY '"' LINES TERMINATED BY '\n';"
curs.execute(import_data)
```

MySQL Workbench

Local instance MySQL80 x

File Edit View Query Database Server Tools Scripting Help



Navigator:

SCHEMAS

Filter objects

- ▶ sakila
- ▶ sys
- ▼ transformer\_chatbot
  - ▼ Tables
    - ▼ chatbot\_table
      - ▶ Columns
      - ▶ Indexes
      - ▶ Foreign Keys
      - ▶ Triggers
    - Views
    - Stored Procedures
    - Functions
  - ▶ world

Query 1 SQL File 3\* chatbot\_script chatbot\_table chatbot\_t

Limit to 1000 rows

1 • SELECT \* FROM transformer\_chatbot.chatbot\_table;

Result Grid Filter Rows: Export: Wrap Cell

	Q	A	
▶	12시 땡!	하루가 또 가네요,	0
	1지망 학교 떨어졌어	위로해 드립니다,	0
	3박4일 놀러가고 싶다	여행은 언제나 좋죠,	0
	3박4일 정도 놀러가고 싶다	여행은 언제나 좋죠,	0
	ppi 싹하네	누살이 찌푸려지죠	

# 전처리 데이터 in

```
# SQL
import pymysql

# SQL작업 끝나면 끝부분에서 curs.close()랑
# curs.commit() 사용하여 작업 내용 SQL서버로 저장

# DB NAME = transformer_chatbot
# TABLE NAME = chatbot_table
# COLUMNS = [question, answer, category]

# MySQL Connection 연결
conn = pymysql.connect(host='localhost', user='root', password='dkdlxl', db='transformer_chatbot', charset='utf8')

# MySQL Connection 연결
conn = pymysql.connect(host='localhost', user='root', password='dkdlxl', db='transformer_chatbot', charset='utf8')

# Connection 으로부터 Cursor 생성
curs = conn.cursor()

# select문으로 데이터 읽기
# sql = "SHOW TABLES;" # use로 되어있는 테이블 확인
sql = "select * from chatbot_table;"

# execute메소드로 위 문장을 SQL 서버로 전송
curs.execute(sql)

# 데이터 Fetch
rows = curs.fetchall()

# read_sql사용하여 train_data에 pandas로 sql문 read해온다.
train_data = pd.read_sql(sql, con=conn)
```

# 하이퍼 파라미터 세팅 & 컴파일, 훈련

```
tf.keras.backend.clear_session()

# Hyper-parameters setting
NUM_LAYERS = 2
D_MODEL = 256
NUM_HEADS = 8
DFF = 512
DROPOUT = 0.1

model = transformer(
    vocab_size=VOCAB_SIZE,
    num_layers=NUM_LAYERS,
    dff=DFF,
    d_model=D_MODEL,
    num_heads=NUM_HEADS,
    dropout=DROPOUT)

MAX_LENGTH = 40
```

```
# 학습률과 옵티마이저를 정의
# 옵티마이저는 adam을 사용
learning_rate = CustomSchedule(D_MODEL)

optimizer = tf.keras.optimizers.Adam(
    learning_rate, beta_1=0.9, beta_2=0.98, epsilon=1e-9)

# 모델 컴파일
model.compile(optimizer=optimizer, loss=loss_function, metrics=[accuracy])

# 총 50회 모델을 학습하겠다.
EPOCHS = 50

model.fit(dataset, epochs=EPOCHS)
```

```
# 학습률과 옵티마이저를 정의
# 옵티마이저는 adam을 사용
learning_rate = CustomSchedule(D_MODEL)

optimizer = tf.keras.optimizers.Adam(
    learning_rate, beta_1=0.9, beta_2=0.98, epsilon=1e-9)

# 모델 컴파일
model.compile(optimizer=optimizer, loss=loss_function, metrics=[accuracy])

# 총 50회 모델을 학습하겠다.
EPOCHS = 50

model.fit(dataset, epochs=EPOCHS)
```



# 예측 및 평가하기

```
# 디코더의 예측 시작
for i in range(MAX_LENGTH):
    predictions = model(inputs=[sentence, output], training=False)

    # 현재(마지막) 시점의 예측 단어를 받아온다.
    predictions = predictions[:, -1:, :]
    # argmax로 가장 높은 신뢰도를 가진 레이블을 찾는다.
    # tf.cast는 텐서를 새로운 자료형으로 변환한다.
    # tf.cast(x, dtype, name=None)
    # x 또는 x.values 를 dtype형으로 변환.
    predicted_id = tf.cast(tf.argmax(predictions, axis=-1), tf.int32)

    # 만약 마지막 시점의 예측 단어가 종료 토큰이라면 예측을 중단한다.
    if tf.equal(predicted_id, END_TOKEN[0]):
        break

    # 마지막 시점의 예측 단어를 출력에 연결한다.
    # 이는 for문을 통해서 디코더의 입력으로 사용될 예정이다.
    output = tf.concat([output, predicted_id], axis=-1)
```

```
def predict(sentence):
    prediction = evaluate(sentence)

    # prediction == 디코더가 리턴한 챗봇의 대답에 해당하는 정수 시퀀스
    # tokenizer.decode()를 통해 정수 시퀀스를 문자열로 디코딩한다.
    predicted_sentence = tokenizer.decode(
        [i for i in prediction if i < tokenizer.vocab_size])

    print('Input: {}'.format(sentence))
    print('Output: {}'.format(predicted_sentence))

    return predicted_sentence
```

```
output = predict('밥 뭐먹을까?')

output = predict("술 먹을까 말까")

output = predict("너무 화가나")

output = predict("게임하고싶은데 할래?")

output = predict("나 너 좋아하는 것 같아")

output = predict("딥 러닝 자연어 처리를 잘 하고 싶어")
```

```
Epoch 46/50
185/185 [=====] - 15s 80ms/step - loss: 0.0064 - accuracy: 0.1735
Epoch 47/50
185/185 [=====] - 15s 78ms/step - loss: 0.0062 - accuracy: 0.1736
Epoch 48/50
185/185 [=====] - 14s 78ms/step - loss: 0.0059 - accuracy: 0.1736
Epoch 49/50
185/185 [=====] - 15s 78ms/step - loss: 0.0060 - accuracy: 0.1736
Epoch 50/50
185/185 [=====] - 15s 79ms/step - loss: 0.0058 - accuracy: 0.1737
Input: 밥 뭐먹을까?
Output: 맛있는 거 드세요 .
Input: 술 먹을까 말까
Output: 좀 먹어도 괜찮아요 .
Input: 너무 화가나
Output: 그럴수록 당신이 힘들 거예요 .
Input: 게임하고싶은데 할래?
Output: 괜찮은 선택이길 바라요 .
Input: 나 너 좋아하는 것 같아
Output: 제가 따라가려면 멀었네요 .
Input: 딥 러닝 자연어 처리를 잘 하고 싶어
Output: 구질구질해도 괜찮아요 .
```

☐ Chapter: 1

☐ Chapter: 2

☐ Chapter: 3

☒ Chapter: 4

Chapter



Things to Improve

개선점



# Using Flask

```
app = Flask(__name__)

# 라인디벨롭 관리자의 발급받은 액세스 키와 토큰은 타인에게 노출
import line_bot_config
line_bot_api = LineBotApi(line_bot_config.CHANNEL_ACCESS_TOKEN)
handler = WebhookHandler(line_bot_config.CHANNEL_SECRET)

@app.route("/callback", methods=['POST'])
def callback():
    # get X-Line-Signature header value
    signature = request.headers['X-Line-Signature']

    @handler.add(MessageEvent, message=TextMessage)
    def handle_message(event):
        line_bot_api.reply_message(
            event.reply_token,
            TextSendMessage(text=event.message.text))
```

```
df = pd.DataFrame.from_dict(doc)


# 한글위키_1 write
out_file1 = './_data/simple-wikipedia-xml-to-csv-master/kowiki111.csv'

# 없으면 만들어서 입력 있으면 있는것에 덧붙여서 입력
if not os.path.exists(out_file1):
    df.to_csv(out_file1, index = False, mode='w',encoding="utf-8-sig")
else:
    df.to_csv(out_file1, index = False, mode='a',encoding="utf-8-sig",header=False)
print("write complete")
```



## 대용량 데이터변환




# Failed Heroku deploying



```
D:\#personal_project\#line_bot>heroku ps:scale web=1 --app transformerchatbot
Scaling dynos... !
! Couldn't find that process type (web).
```


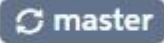
 **HEROKU**

Jump to Favorites, Apps, Pipelines, Spaces...



 < >  transformerchatbot 

 Open app  More <



 younggi0123/transformerchatbot  master

[Overview](#) [Resources](#) [Deploy](#) [Metrics](#) [Activity](#) [Access](#) [Settings](#)

Installed add-ons **\$0.00/month** [Configure Add-ons](#) <

Latest activity [All Activity](#) <

There are no add-ons for this app  
You can add add-ons to this app and they will

 **younggi0123@naver.com: Build failed**  
Yesterday at 8:31 PM · [View build log](#)



# Failed to connect line

**LINE Developers** [News](#) [Products](#) [Documentation](#) [FAQ](#) [Glossary](#) [Community](#)

**Console home**

**Providers**


**Admin**

고나리소프트

**Tools**

**Support**


TOP > [고나리소프트](#) > [고나리1](#) > **Messaging API**

 **고나리1** Admin Messaging API


[Basic settings](#) [Messaging API](#) [LIFF](#) [Security](#)

**Messaging API settings**

**Bot information**


**Bot basic ID** @hhj4941f 

**QR code**



읽음  
오후 8:40

ㅍㅍ



메시지 감사합니다 😊


죄송합니다만 이 계정으로  
개별적으로 답장을 드리지  
못합니다 😞

다음 소식을 기대해주세요 ✨

오후 8:40

읽음  
오후 8:52

하...




메시지 감사합니다 😊


죄송합니다만 이 계정으로  
개별적으로 답장을 드리지  
못합니다 😞


다음 소식을 기대해주세요 ✨


오후 8:52

+









위키독스attention is all you need - GOOGLE

위키독스<https://wikidocs.net> 점프투 파이썬, 딥러닝을 이용한 자연어처리  
전창욱 외(2020), 텐서플로2와 머신러닝으로 시작하는 자연어처리  
구글 BERT의 정석\_수다르산 라비찬디란

[튜토리얼4] 언어 이해를 위한 트랜스포머(Transformer) 모델  
<https://url.kr/48wkih>

트랜스포머(TRANSFORMER)  
[https://yngie-c.github.io/nlp/2020/07/01/nlp\\_transformer/](https://yngie-c.github.io/nlp/2020/07/01/nlp_transformer/)

[머신러닝/딥러닝] 10-2. Attention in RNN-Encoder-Decoder  
<https://sonsnotation.blogspot.com/2020/11/11-attention-transformer-models.html>

자연어처리 발전흐름(RNN에서 BERT)  
<https://han-py.tistory.com/249>  
Transformer등장배경  
[https://blog.naver.com/jaeyoon\\_95/221760816958](https://blog.naver.com/jaeyoon_95/221760816958)  
유튜브

토큰on세미나 python과 tensorflow를 활용한 ai 챗봇 - YouTube

트랜스포머 (어텐션 이즈 올 유 니드) - YouTube

텍스트 인식과 머신러닝으로 인공지능(AI) 감정 만들기 (ft. ZenBook Duo UX481) - YouTube

Transformer Neural Networks - EXPLAINED! (Attention is all you need) - YouTube

Python과 Tensorflow를 활용한 AI 챗봇 개발 3강

[딥러닝 기계 번역] Transformer- Attention Is All You Need (꼼꼼한 딥러닝 논문 리뷰와 코드 실습) - YouTube

[Attention Is All You Need] - Transformer 트랜스포머 소개 - YouTube



Q&A