

자연어 처리

# 주요 내용

- 텍스트 벡터화
- 단어 임베딩
- 트랜스포머 아키텍처
- 시퀀스-투-시퀀스 학습
- 영어-한국어 번역

# 자연어 처리 소개

- 컴퓨터 프로그래밍 언어: 파이썬, 자바, C, C++, C#, 자바스크립트 등
- 자연어: 일상에서 사용되는 한국어, 영어 등
- 자연어의 특성상 정확한 분석을 위한 알고리즘 구현 매우 어려움
- 딥러닝 기법이 활용 이전: 별로 성공적이지 못함

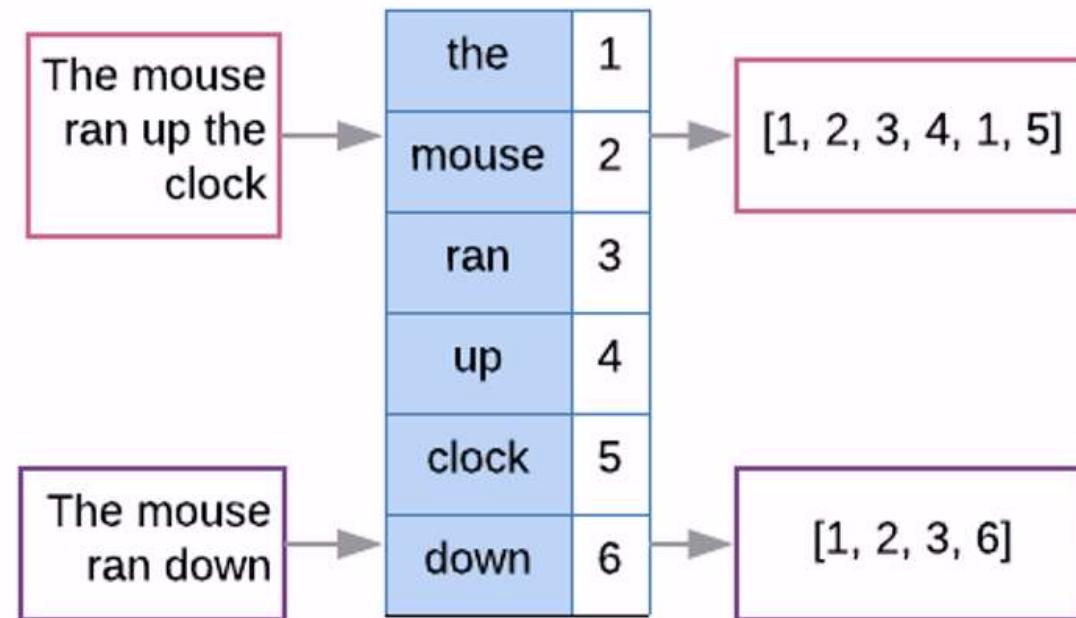
## 자연어 처리

- 1990년대: 언어의 이해가 아닌 통계적으로 유용한 정보를 예측 연구 진행
- 텍스트 분류: "이 텍스트의 주제는?"
- 내용 필터링: "욕설 사용?"
- 감성 분석: "그래서 내용이 긍정이야 부정이야?"
- 언어 모델링: "이 문장 다음에 어떤 단어가 와야 할까?"
- 번역: "이거를 한국어로 어떻게 말해?"
- 요약: "이 기사를 한 줄로 요약해 볼래?"

## 머신러닝 활용

- 1990 - 2010년대 초반: 결정트리, 로지스틱 회귀 모델이 주로 활용됨.
- 2014-2015: LSTM 등 시퀀스 처리 알고리즘 활용 시작
- 2015-2017: (양방향) 순환신경망이 기본적으로 활용됨.
- 2017-2018: 트랜스포머 transformer 아키텍처가 많은 난제들을 해결함.
- 2022 이후: 트랜스포머 아키텍처를 이용한 GPT의 혁명적 발전

# 텍스트 벡터화



## 텍스트 벡터화 단계

- **텍스트 표준화**: text standardization: 소문자화, 마침표 제거 등등
- **토큰화**: tokenization: 기본 단위의 유닛 units으로 쪼개기. 문자, 단어, 단어 집합 등이 토큰으로 활용됨.
- **어휘 색인화**: vocabulary indexing: 토큰 각각을 하나의 정수 색인(인덱스)으로 변환.

## 텍스트 표준화

- 모두 소문자화
- ., ;, ?, ' 등 특수 기호 제거
- 특수 알파벳 변환:
  - "é"를 "e"로,
  - "æ"를 "ae"로 등등
- 동사/명사의 기본형 활용:
  - "cats"를 "[cat]"로,
  - "was staring"과 "stared"를 "[stare]"로 등등.

다음 두 텍스트를 표준화를 통해 동일한 텍스트로 변환 가능

```
"sunset came. i was staring at the Mexico sky. Isn't nature splendid??"
```

```
"Sunset came; I stared at the M&acute;xico sky. Isn't nature splendid?"
```

표준화 결과

```
"sunset came i [stare] at the mexico sky isnt nature splendid"
```

## 토큰화

- 단어 기준 토큰화
  - 공백으로 구분된 단어들로 쪼개기.
  - 경우에 따라 동사 어근과 어미를 구분하기도 함: "star+ing", "call+ed" 등등
- N-그램 토큰화
  - N-그램 토큰: 연속으로 위치한 N 개(이하)의 단어 묶음
  - 예제: "the cat", "he was" 등은 2-그램(바이그램) 토큰
- 문자 기준 토큰화
  - 하나의 문자를 하나의 토큰으로 지정.
  - 텍스트 생성, 음성 인식 등에서 활용됨.

# 단어 주머니

"the cat sat on the mat." 에 대한 바이그램과 3-그램 주머니

- 2-그램(바이그램) 주머니

```
{"the", "the cat", "cat", "cat sat",
"sat",
"sat on", "on", "on the", "the mat",
"mat"}
```

- 3-그램 주머니

```
{"the", "the cat", "cat", "cat sat", "the cat
sat",
"sat", "sat on", "on", "cat sat on", "on the",
"sat on the", "the mat", "mat", "on the mat"}
```

## 어휘 색인화

- 훈련셋에 포함된 모든 토큰들의 색인(인덱스) 생성
- 보통 사용 빈도가 높은 1만, 2만, 또는 3만 개의 단어만 대상

```
from tensorflow.keras.datasets import imdb  
imdb.load_data(num_words=10000)
```

- 텍스트를 단어 색인으로 구성된 리스트로 변환

```
[1, 14, 22, 16, 43, 530, 973, 1622, 1385, 65, 0, 0, 0, 0, 0]
```

## TextVectorization 층

```
from tensorflow.keras.layers import TextVectorization

text_vectorization = TextVectorization(
    standardize='lower_and_strip_punctuation', # 기본값
    split='whitespace', # 기본값
    ngrams=None, # 기본값
    output_mode='int', # 기본값
)
```

## 텍스트 벡터화 예제

```
>>> dataset = [  
...     "I write, erase, rewrite",  
...     "Erase again, and then",  
...     "A poppy blooms.",  
... ]
```

어휘 색인은 `adapt()` 메서드를 이용하여 만든다.

```
>>> text_vectorization.adapt(dataset)
```

생성된 어휘 색인은 다음과 같다.

```
>>> vocabulary = text_vectorization.get_vocabulary()
>>> vocabulary
[ ' ',  
  '[UNK]',  
  'erase',  
  'write',  
  'then',  
  'rewrite',  
  'poppy',  
  'i',  
  'blooms',  
  'and',  
  'again',  
  'a' ]
```

생성된 어휘 색인을 활용하여 새로운 텍스트를 벡터화 해보자.

```
>>> test_sentence = "I write, rewrite, and still rewrite again"  
>>> encoded_sentence = text_vectorization(test_sentence)  
>>> print(encoded_sentence)  
tf.Tensor([ 7  3  5  9  1  5 10], shape=(7,), dtype=int64)
```

벡터화된 텐서로부터 텍스트를 복원하면 표준화된 텍스트가 생성된다.

```
>>> inverse_vocab = dict(enumerate(vocabulary))
>>> decoded_sentence = " ".join(inverse_vocab[int(i)] for i in
encoded_sentence)
>>> print(decoded_sentence)
i write rewrite and [UNK] rewrite again
```

IMDB 영화 후기 데이터셋 벡터화

## 과정 1: 데이터셋 다운로드 후 압축 풀기

```
aclImdb/  
...test/  
.....pos/  
.....neg/  
...train/  
.....pos/  
.....neg/
```

## 과정 2: 검증셋 준비

```
aclImdb/  
...test/  
.....pos/  
.....neg/  
...train/  
.....pos/  
.....neg/  
...val/  
.....pos/  
.....neg/
```

### 과정 3: 텐서 데이터셋 준비

## 과정 4: 텍스트 벡터화

```
max_length = 600    # 후기 길이 제한  
max_tokens = 20000  # 단어 사용빈도 제한  
  
text_vectorization = layers.TextVectorization(  
    max_tokens=max_tokens,  
    output_mode="int",  
    output_sequence_length=max_length,  
)
```

## 어휘 색인화

```
# 어휘 색인 생성 대상 훈련셋 후기 텍스트 데이터셋
text_only_train_ds = train_ds.map(lambda x, y: x)

# 어휘 색인
text_vectorization.adapt(text_only_train_ds)
```

## 데이터셋 벡터화

```
# 후기를 길이가 2만인 정수들의 리스트로 변환
int_train_ds = train_ds.map(lambda x, y: (text_vectorization(x), y))

int_val_ds = val_ds.map(lambda x, y: (text_vectorization(x), y))

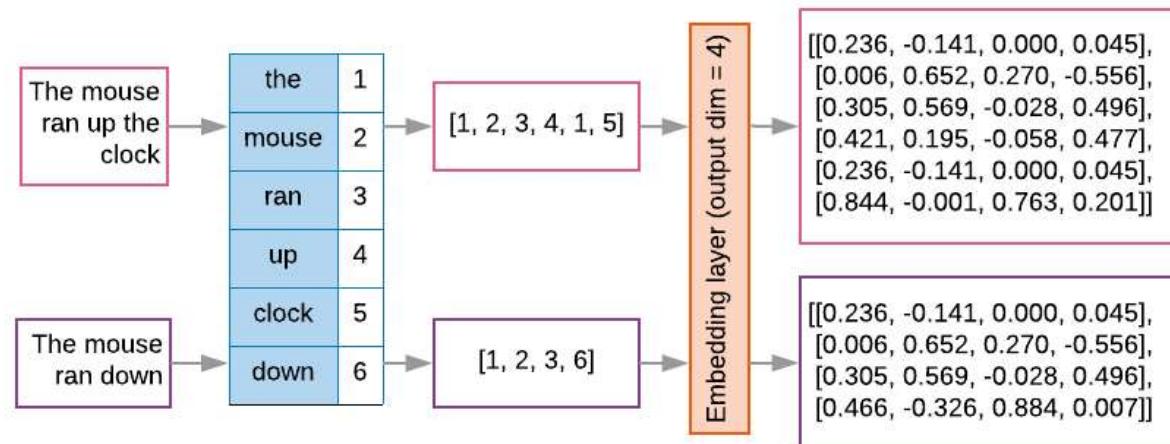
int_test_ds = test_ds.map(lambda x, y: (text_vectorization(x), y))
```



# 단어 임베딩

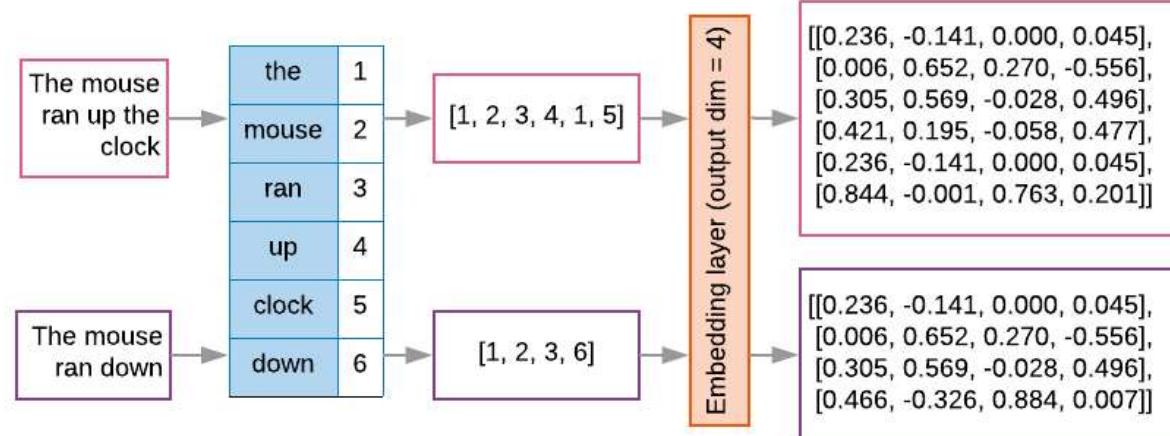
```
vocab_size = 7 # 총 어휘 수  
embed_dim = 4 # 단어별로 4 개의 특성 파악.
```

```
embedding_layer = ayers.Embedding(input_dim=vocab_size,  
                                    output_dim=embed_dim)
```

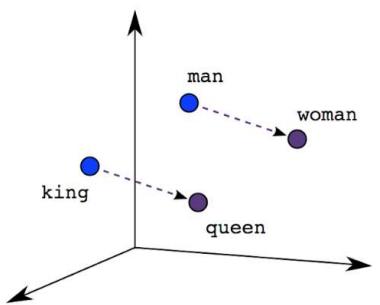


## 임베딩 행렬

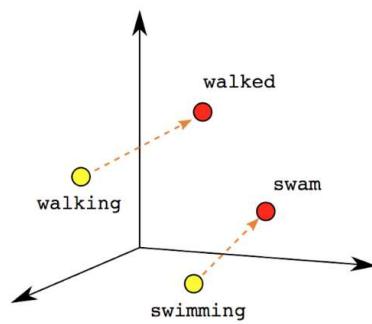
```
embedding_matrix = [[-0.012, 0.005, 0.008, 0.001],  
[0.236, -0.141, 0.000, 0.045],  
[0.006, 0.652, 0.270, -0.556],  
[0.305, 0.569, -0.028, 0.496],  
[0.421, 0.195, -0.058, 0.477],  
[0.844, -0.001, 0.763, 0.201],  
[0.466, -0.326, 0.884, 0.007]]
```



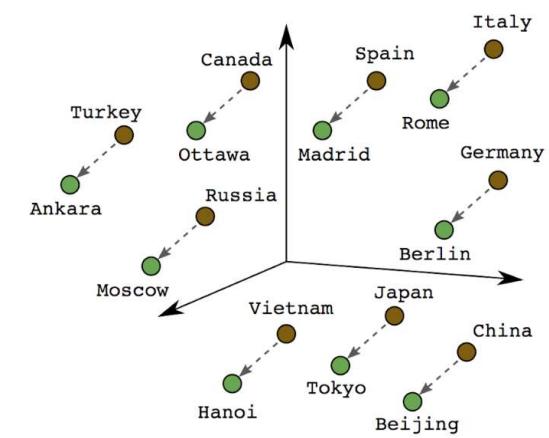
## 단어 임베딩의 의미



Male-Female



Verb Tense



Country-Capital

## 단어 임베딩 활용법

```
vocab_size = 20000 # 총 어휘 수
embed_dim = 256 ... # 단어별로 256 개의 특성 파악.

inputs = keras.Input(shape=(None,), dtype="int64")

# 단어 임베딩 실행
x = layers.Embedding(vocab_size, embed_dim)(inputs)

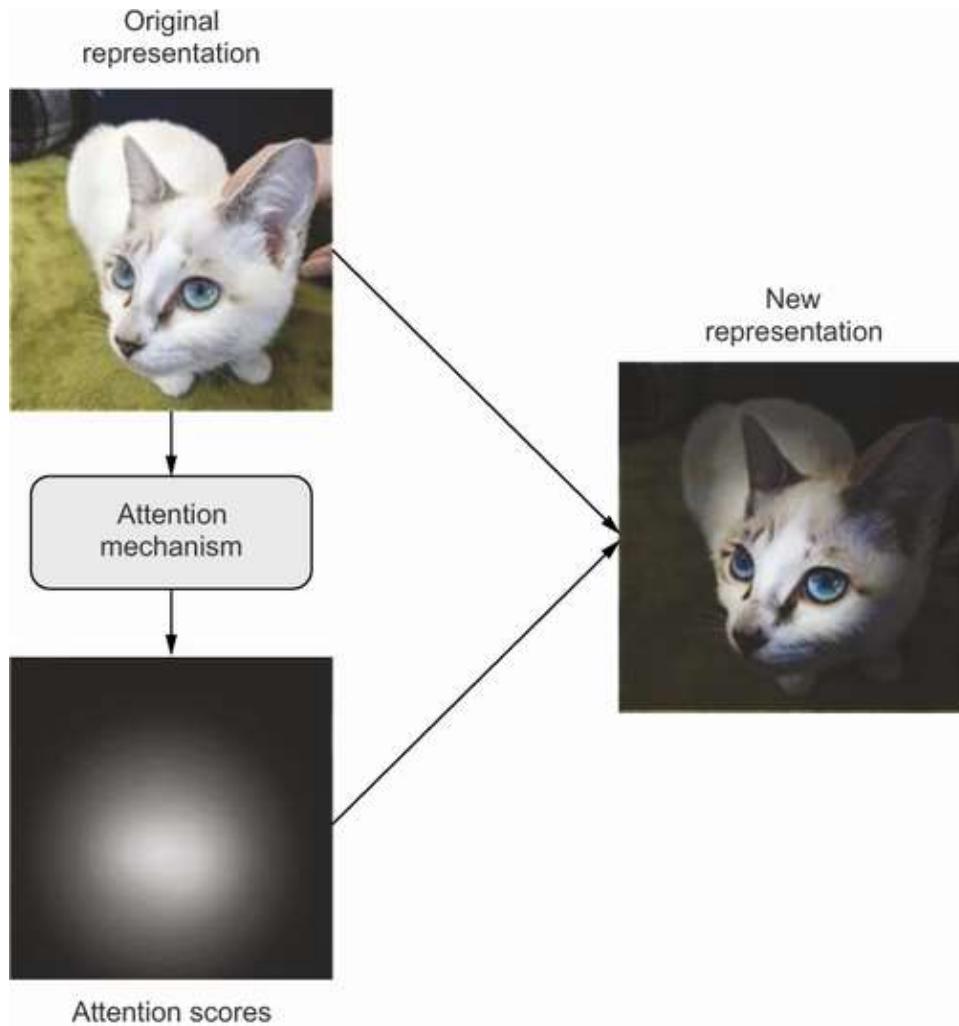
# 트랜스포머 인코더: 텍스트 내용 파악
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)

# 출력 층
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.5)(x)
outputs = layers.Dense(1, activation="sigmoid")(x)

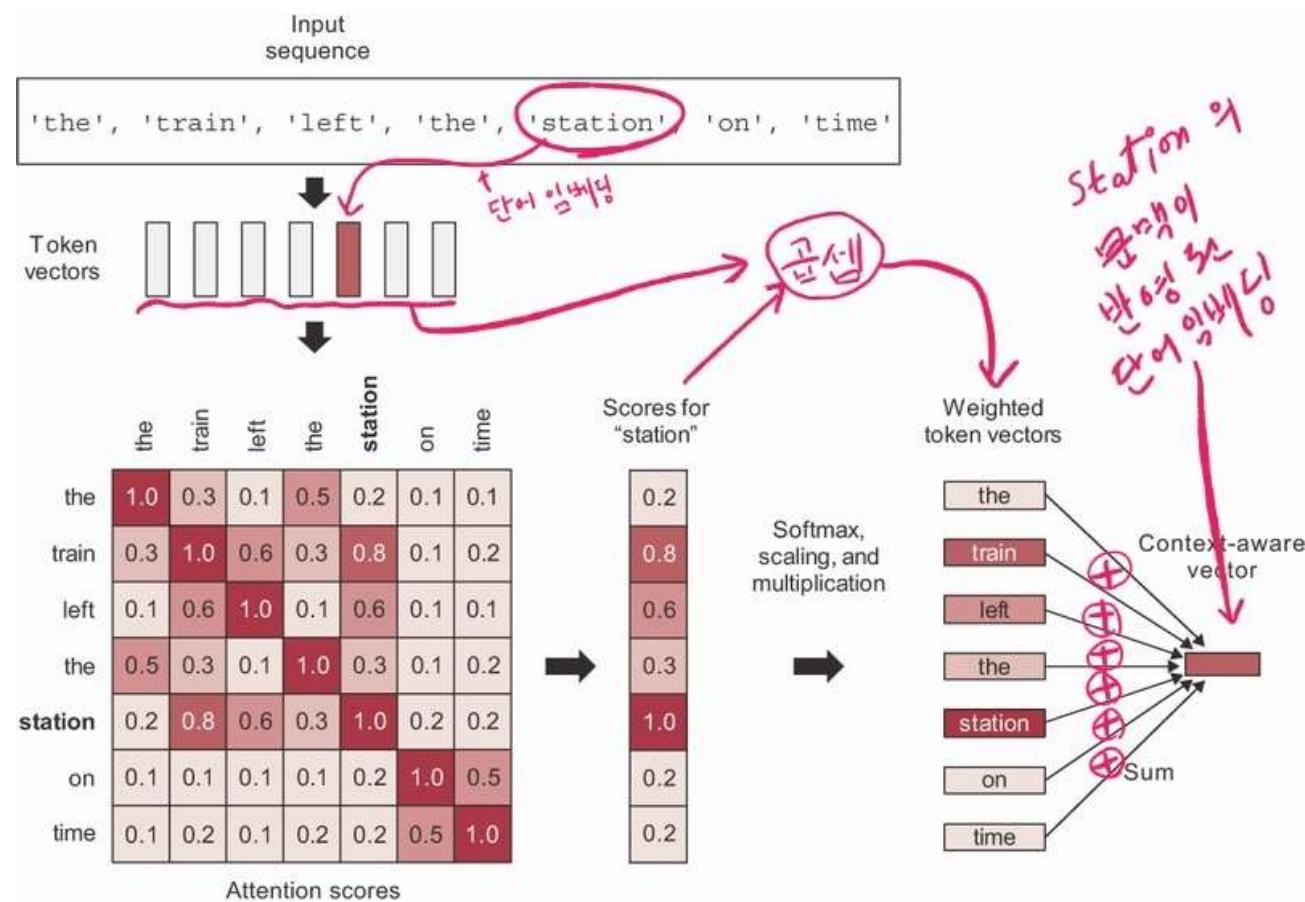
model = keras.Model(inputs, outputs)
```

트랜스포머 아키텍처

# 어텐션



## 자연어 처리에서의 셀프 어텐션



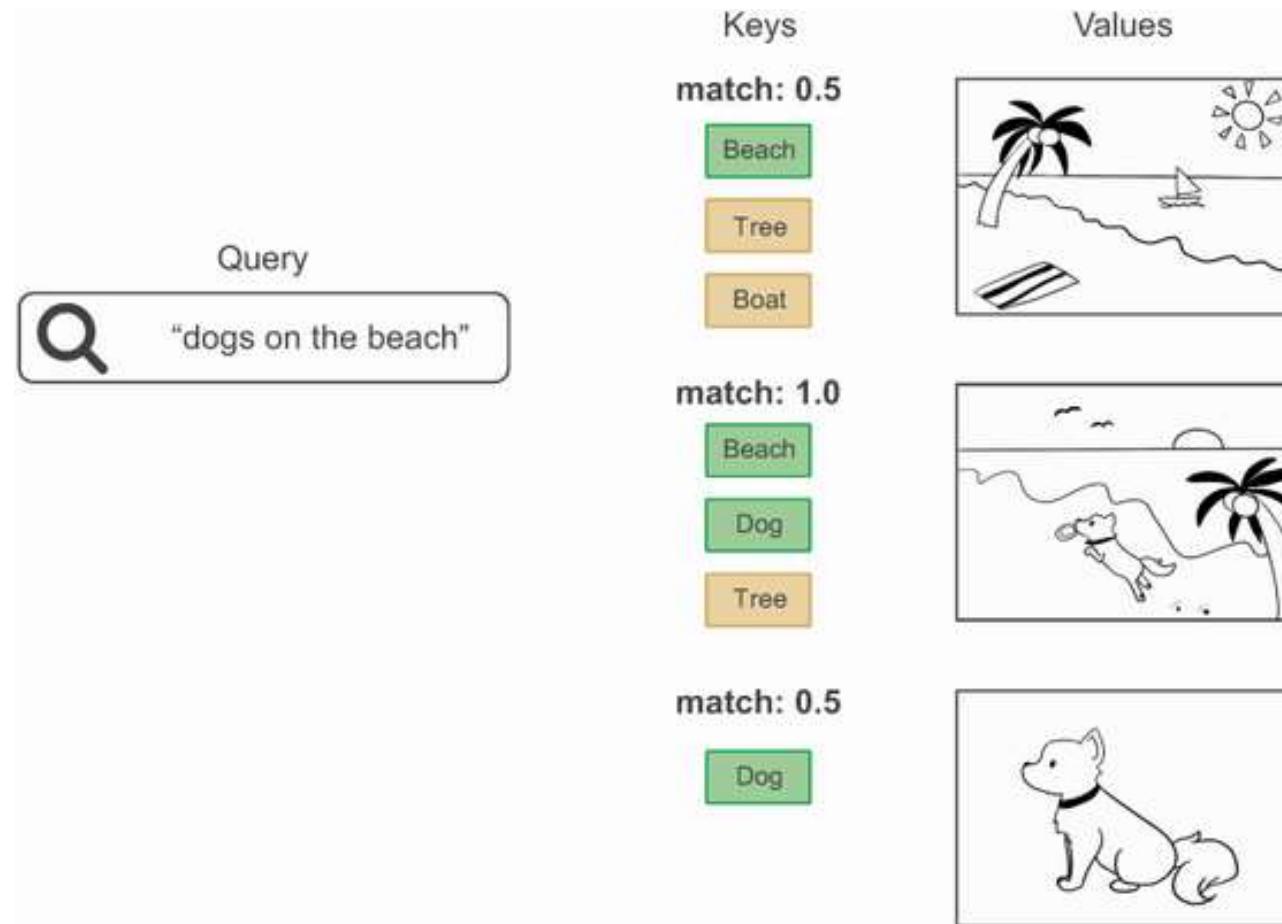
## 케라스 MultiHeadAttention 층

```
num_heads = 2 # 두 개의 셀프 어텐션 동시 진행. 각각 다른 문맥을 파악.  
embed_dim = 256 # 단어 임베딩된 벡터의 길이
```

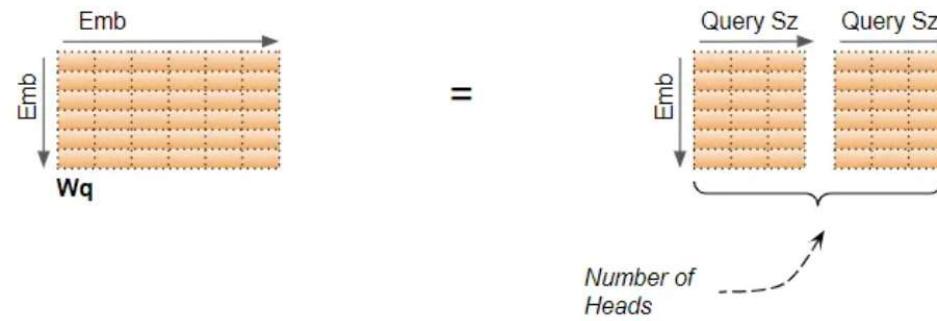
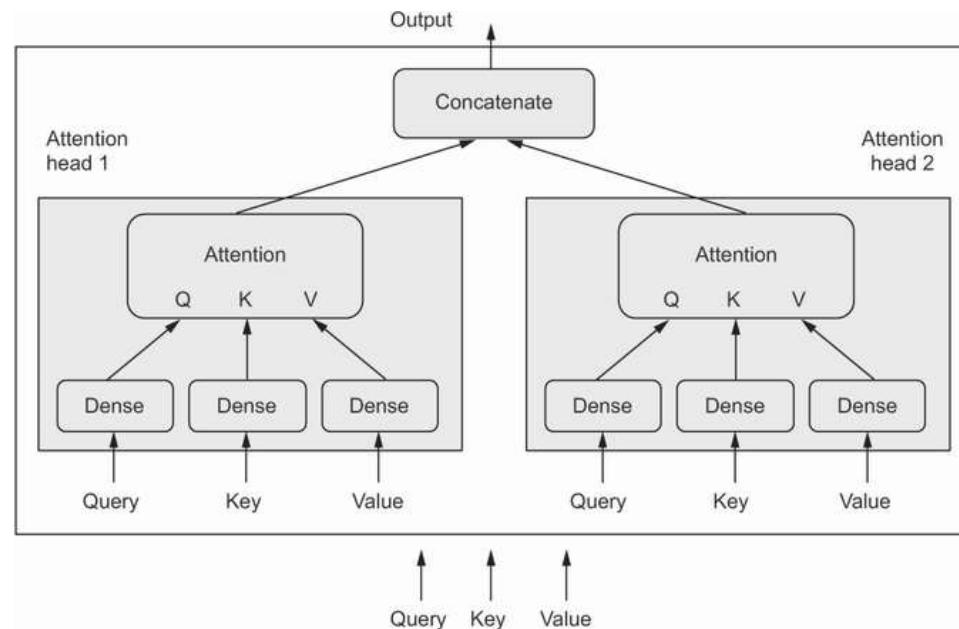
```
mha_layer = MultiHeadAttention(num_heads=num_heads, key_dim=embed_dim)  
outputs = mha_layer(inputs, inputs, inputs)
```

# 질문-열쇠-값

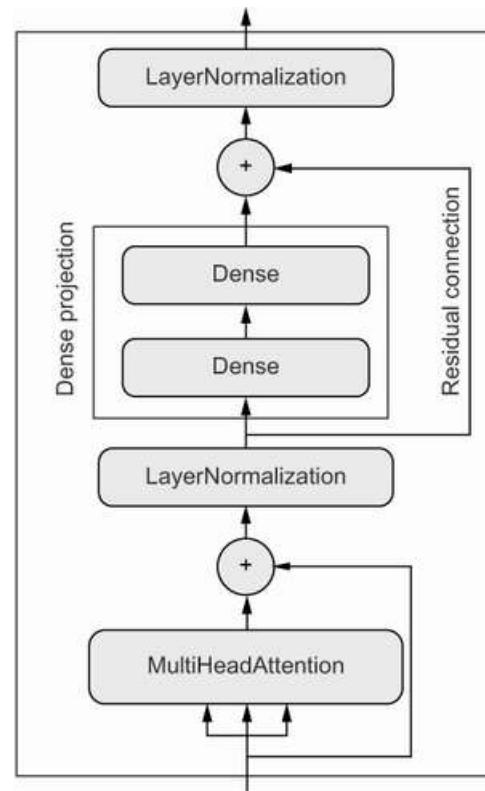
query-key-value



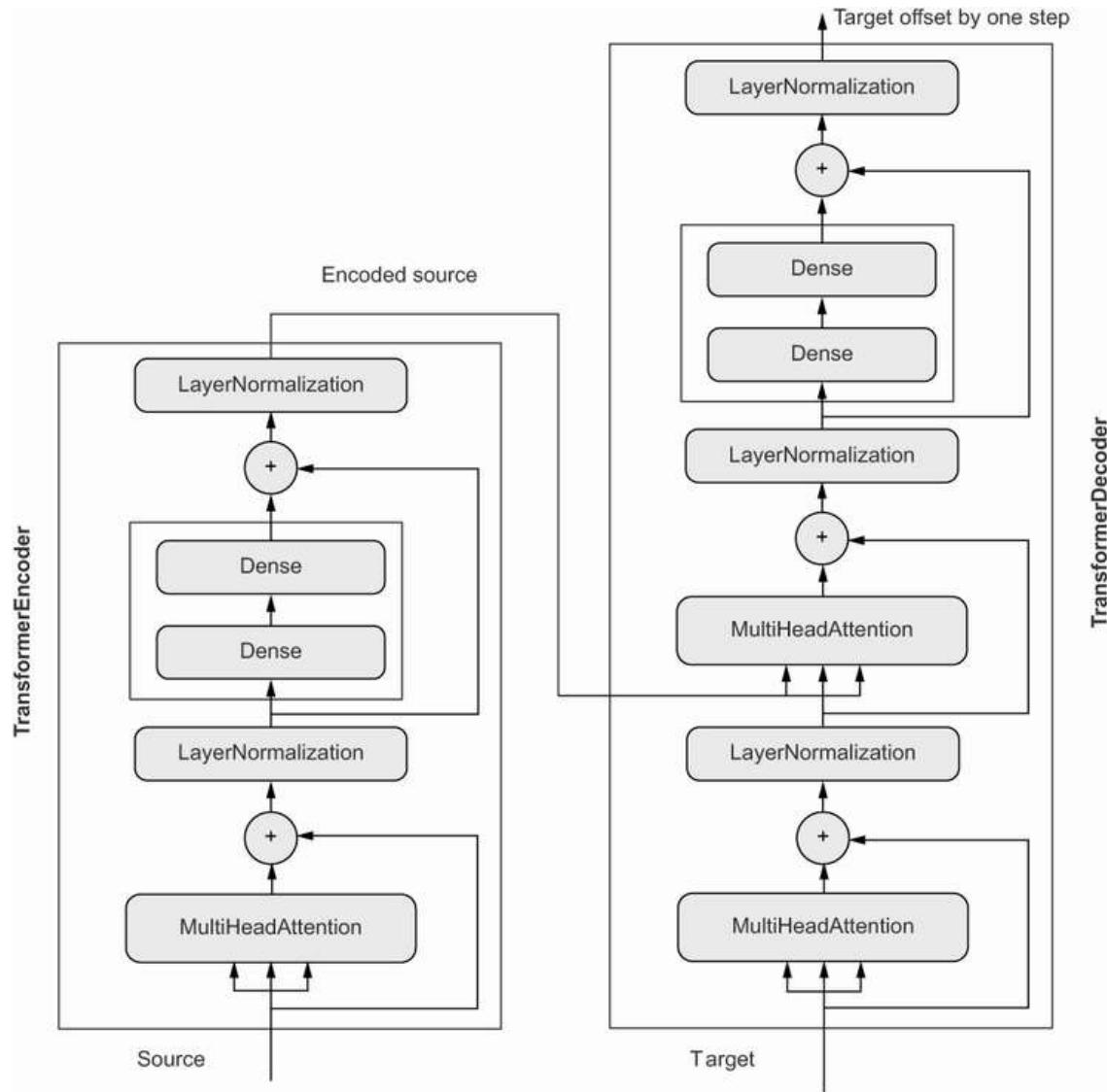
# 멀티헤드 어텐션



# 트랜스포머 인코더



# 트랜스포머 아키텍처



# 트랜스포머 인코더 구현

```
class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        super().__init__(**kwargs)
        self.embed_dim = embed_dim # 예를 들어 256
        self.dense_dim = dense_dim # 예를 들어 32
        self.num_heads = num_heads # 예를 들어 2

        # 어텐션 층 지정
        self.attention = layers.MultiHeadAttention(
            num_heads=num_heads, key_dim=embed_dim)
        # 밀집층 블록 지정
        self.dense_proj = keras.Sequential(
            [layers.Dense(dense_dim, activation="relu"),
             layers.Dense(embed_dim),])
        # 층 정규화 지정
        self.layernorm_1 = layers.LayerNormalization()
        self.layernorm_2 = layers.LayerNormalization()

        # 트랜스포머 인코더의 순전파
    def call(self, inputs, mask=None):
        ...
```

```
class TransformerEncoder(layers.Layer):
    def __init__(self, embed_dim, dense_dim, num_heads, **kwargs):
        ...
        ...

    # 트랜스포머 인코더의 순전파
    def call(self, inputs, mask=None):
        if mask is not None:
            mask = mask[:, tf.newaxis, :]
        attention_output = self.attention(
            inputs, inputs, attention_mask=mask)
        proj_input = self.layernorm_1(inputs + attention_output)
        proj_output = self.dense_proj(proj_input)
        return self.layernorm_2(proj_input + proj_output)

    # 트랜스포머 인코더의 속성 지정
    # 훈련된 모델을 저장할 때 활용됨
    def get_config(self):
        config = super().get_config()
        config.update({
            "embed_dim": self.embed_dim,
            "num_heads": self.num_heads,
            "dense_dim": self.dense_dim,
        })
        return config
```

## 트랜스포머 인코더 활용

```
vocab_size = 20000
embed_dim = 256
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")

x = layers.Embedding(vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)

# (600, 256) 모양의 단어 임베딩된 텍스트 텐서를
# 길이가 256인 1차원 어레이로 변환.
x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.5)(x)

outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs, outputs)
```

## 단어 위치 임베딩

```
class PositionalEmbedding(layers.Layer):
    def __init__(self, sequence_length, input_dim, output_dim, **kwargs):
        super().__init__(**kwargs)
        self.token_embeddings = layers.Embedding(
            input_dim=input_dim, output_dim=output_dim)
        self.position_embeddings = layers.Embedding(
            input_dim=sequence_length, output_dim=output_dim)
        self.sequence_length = sequence_length
        self.input_dim = input_dim
        self.output_dim = output_dim

    def call(self, inputs):
        length = tf.shape(inputs)[-1] # 문장의 단어 수.
        positions = tf.range(start=0, limit=length, delta=1)
        embedded_tokens = self.token_embeddings(inputs)
        embedded_positions = self.position_embeddings(positions)
        return embedded_tokens + embedded_positions

    ...
```

## 단어위치 인식 트랜스포머 인코더

```
vocab_size = 20000
sequence_length = 600
embed_dim = 256
num_heads = 2
dense_dim = 32

inputs = keras.Input(shape=(None,), dtype="int64")

x = PositionalEmbedding(sequence_length, vocab_size, embed_dim)(inputs)
x = TransformerEncoder(embed_dim, dense_dim, num_heads)(x)

x = layers.GlobalMaxPooling1D()(x)
x = layers.Dropout(0.5)(x)

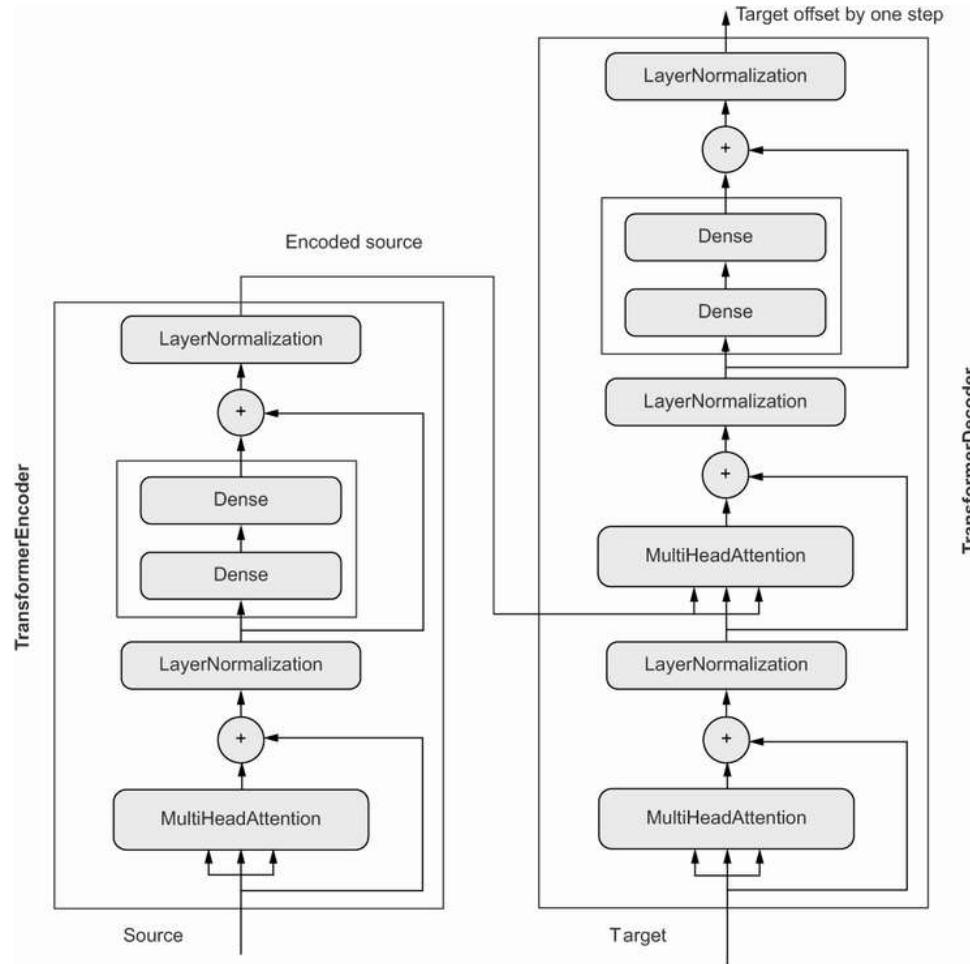
outputs = layers.Dense(1, activation="sigmoid")(x)

model = keras.Model(inputs, outputs)
```

# 시퀀스-투-시퀀스 학습: 기계 번역

- 기계 번역
- 텍스트 내용 요약
- 질문 답변
- 챗봇 기능
- 텍스트 생성

# 트랜스포머 아키텍처: 기계 번역



- 훈련에서의 출력값: 길이가 20인 문장이 들어오면 그 다음에 이어질 단어가 포함된 길이가 20인 단어로 구성된 문장
- 실전 활용에서의 출력값: 길이가 20인 문장이 들어오면 문장을 구성할 단어를 하나씩 생성

