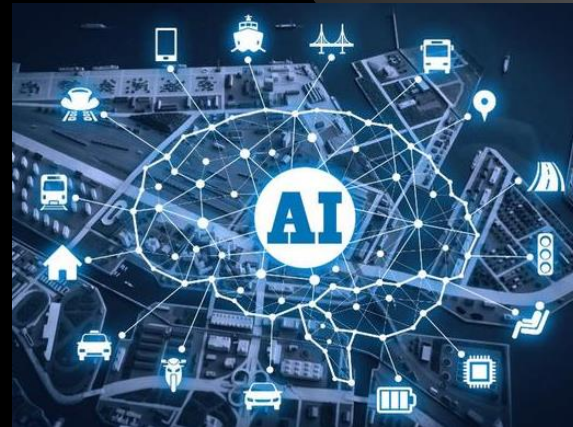


RNN, LSTM, Seq2Seq, Transformer



이 슬라이드에서 사용한 서체 :

- Open Sans(<https://ko.cooltext.com/Download-Font-Open+Sans>)
- KoPubWorld돋움체(<http://www.kopus.org/biz/electronic/font.aspx>)



1장. 자연어처리 인공지능경망

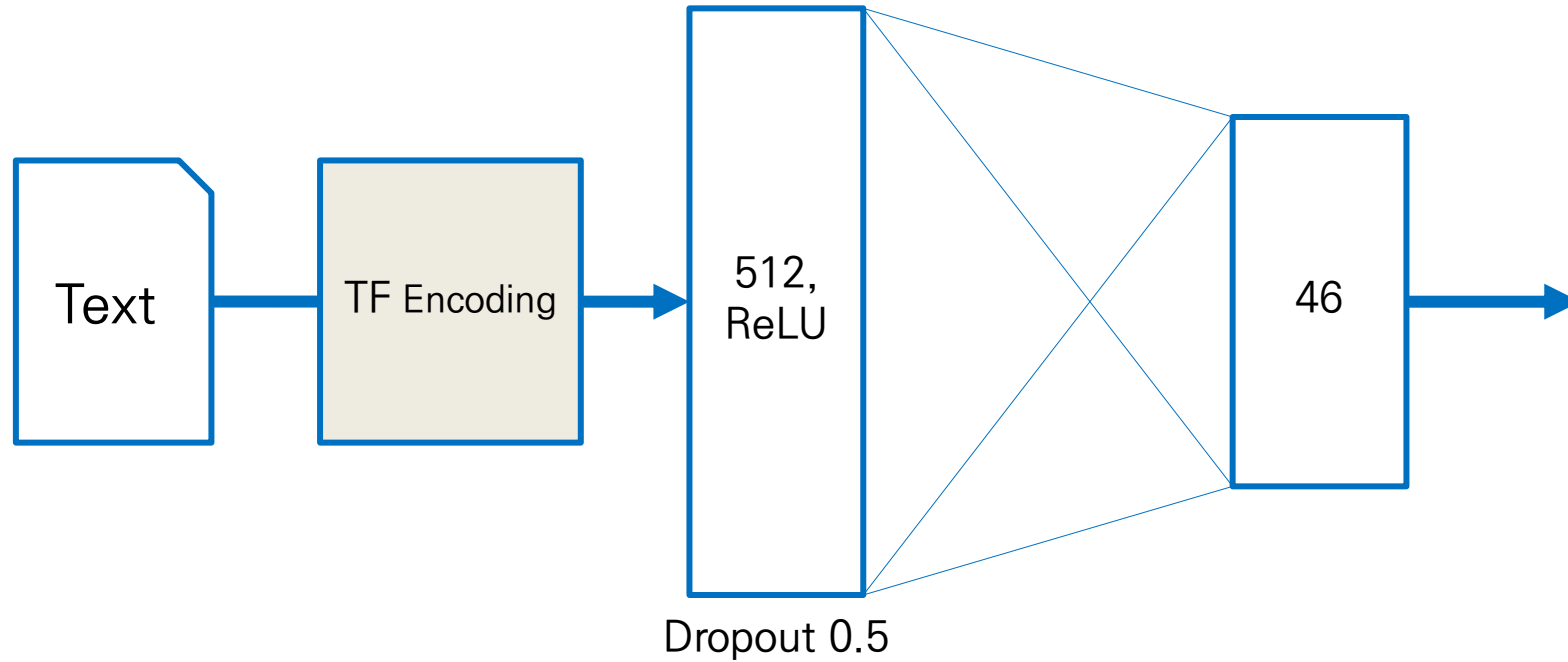
1절. 순환신경망

1장. 자연어처리 인공지능망



DNN을 기억하시나요?

1장. 자연어처리 인공지능 / 1절. 순환신경망



순환신경망 개요

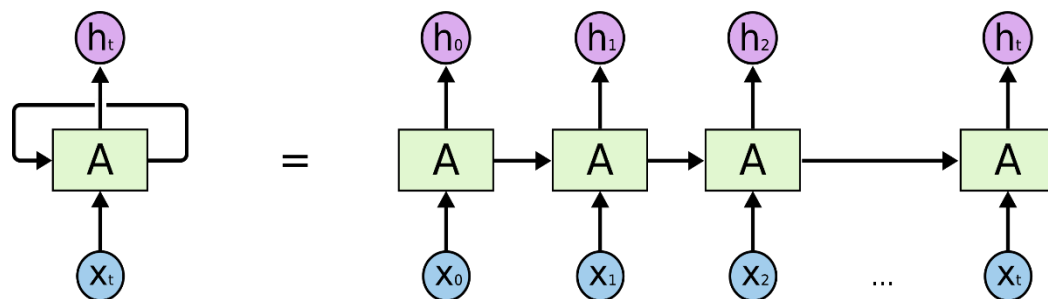
1장. 자연어처리 인공지능망 / 1절. 순환신경망

순환신경망은 1986년 데이비드 루멜하트(David Rumelhart)가 개발한 알고리즘

순환신경망은 시계열 데이터를 학습하는 딥러닝 기술

순환신경망은 기준 시점(t)과 다음 시점($t+1$)에 네트워크를 연결함

순환신경망은 AI 번역, 음성인식, 주가 예측의 대표적 기술임



RNN 종류

1장. 자연어처리 인공지능망 / 1절. 순환신경망

One to Many : 입력이 하나이고 출력이 여러 개 생성

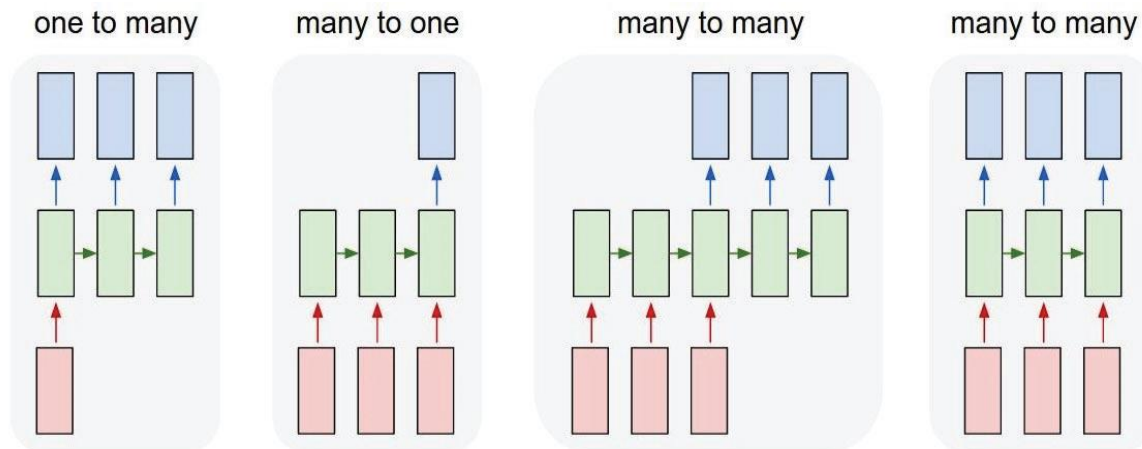
- ▶ One to Many는 영상에 캡션을 달아야 할 때 사용

Many to One : 여러 입력이며, 하나의 출력을 생성

- ▶ 영화평을 분류할 경우 Many to One이 사용

Many to Many : 여러 입력이며, 출력도 여러 개 생성

- ▶ 3번째 Many to Many 구조는 번역에 사용



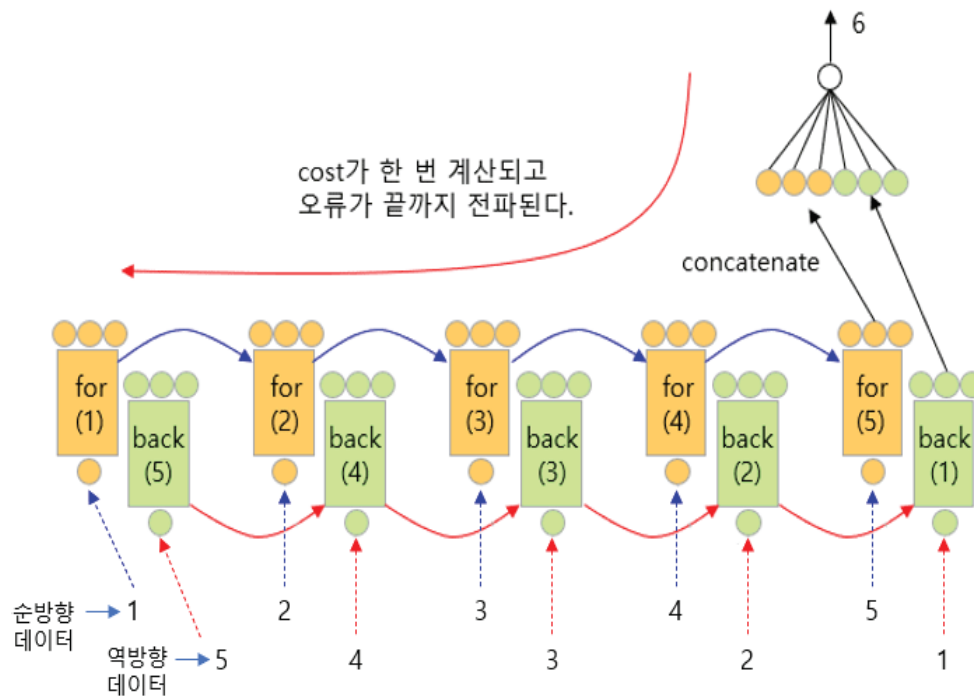
Bidirectional RNN

1장. 자연어처리 인공지능망 / 1절. 순환신경망

양방향 RNN은 두 개의 RNN을 하나로 묶음

하나는 순방향으로 가중치를 수정하며, 다른 하나는 역방향으로 가중치를 수정함

- ▶ RNN 단어가 길면 과거의 일을 잊는 것을 방지하고 또한 미래의 사실을 과거에 반영할 수 있는 구조



2절. LSTM과 GRU

1장. 자연어처리 인공지능망

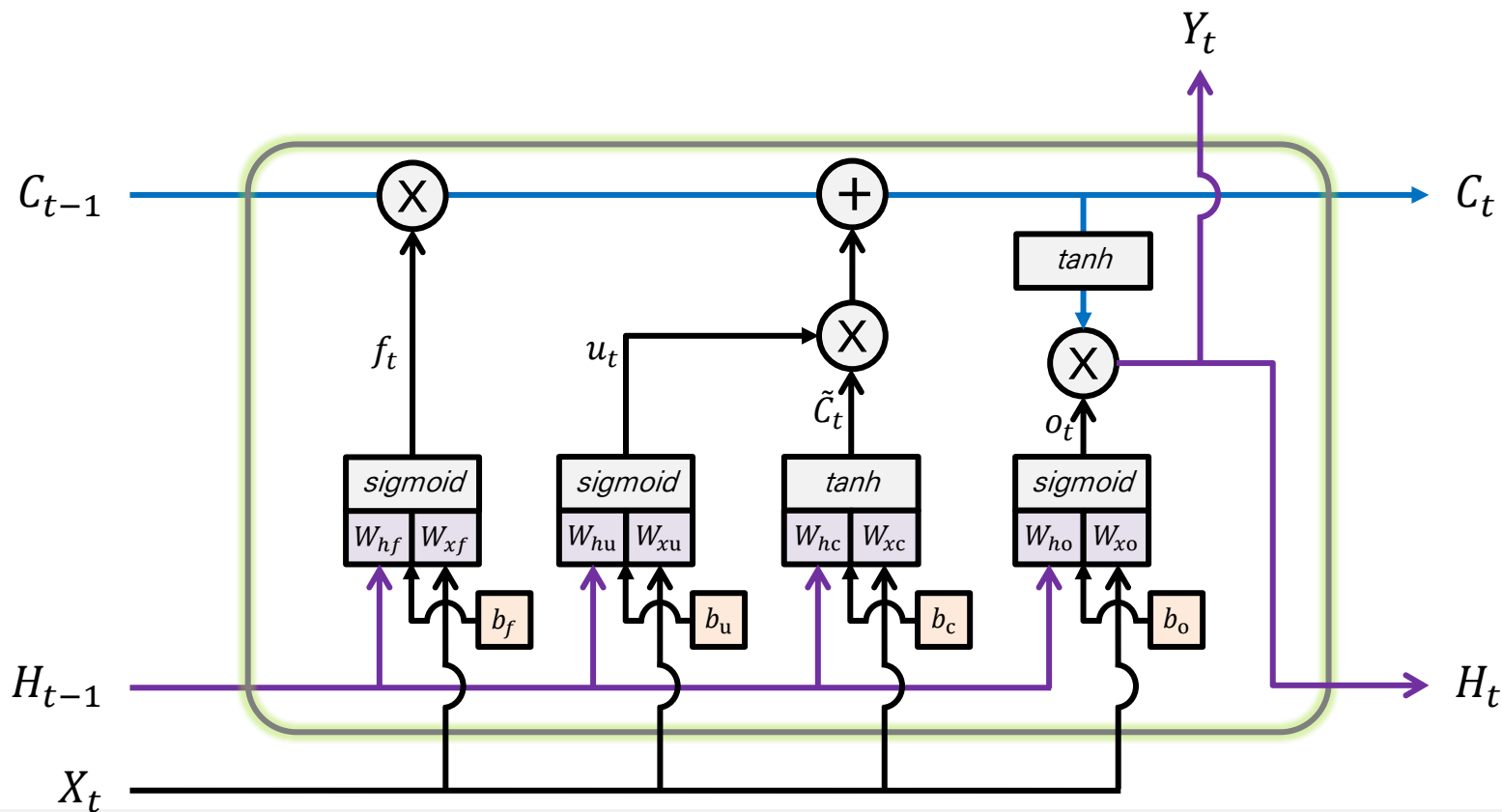


LSTM

1장. 자연어처리 인공지능망 / 2절. LSTM과 GRU

RNN의 단기 기억(Short-Term Memory) 문제를 해결하기 위해 만들어진 것

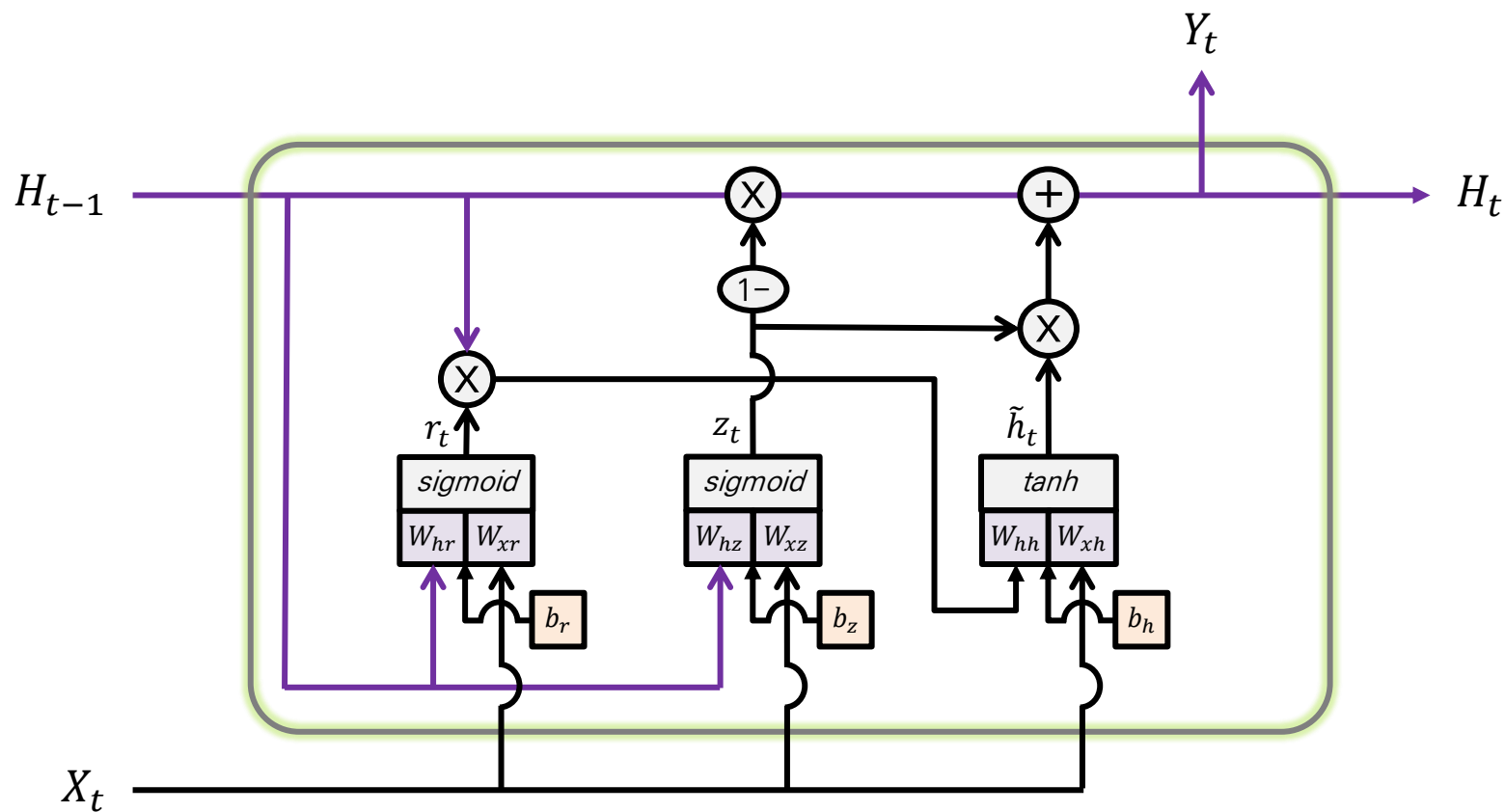
최근의 기억은 유지하지만 오래된 기억은 전달되지 않는 문제를 해결하고자 하는 것



GRU

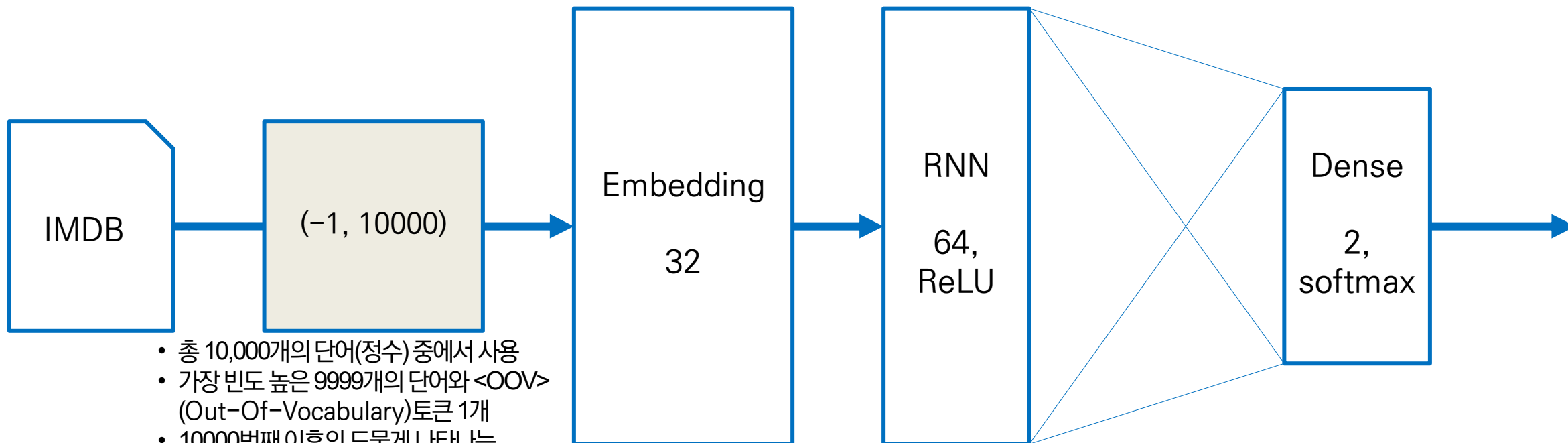
1장. 자연어처리 인공지능망 / 2절. LSTM과 GRU

GRU(Gated Recurrent Unit): LSTM의 게이트를 단순화시켜 속도를 빠르게 개선한 것



실습 - LSTM으로 영화평 분류하기 모델 구조

1장. 자연어처리 인공지능망

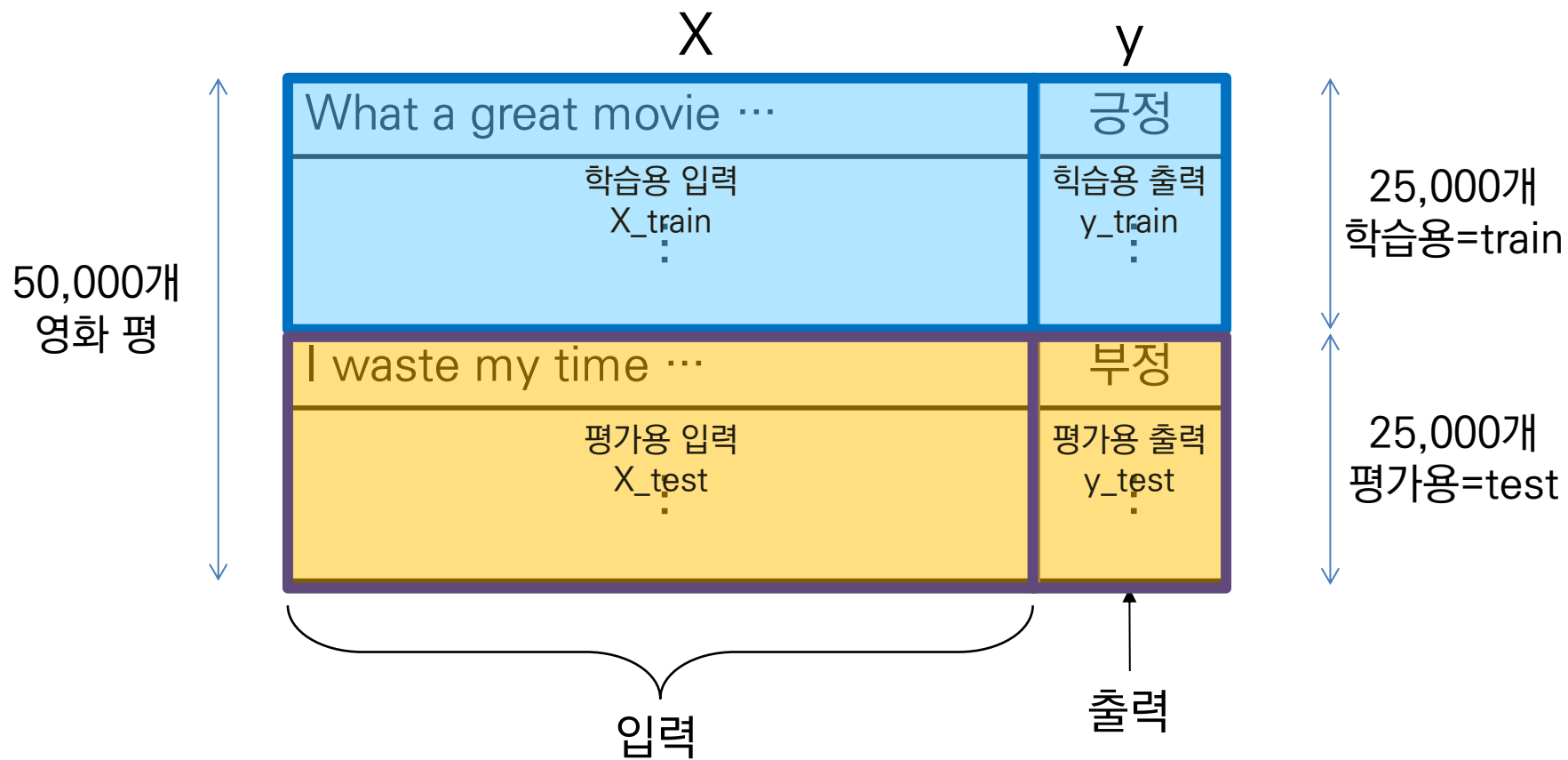


- 총 10,000개의 단어(정수) 중에서 사용
- 가장 빈도 높은 9999개의 단어와 <OOV> (Out-Of-Vocabulary)토큰 1개
- 10000번째 이후의 드물게 나타나는 단어는 모두 <OOV>로 처리됩니다.
- 이 <OOV>는 정수로 매핑되며, 보통 인덱스 1에 위치합니다 (0은 보통 padding용)
- 그런데, IMDB 데이터셋은 학습 편의를 위해 앞의 인덱스 0~3을 특수 목적으로 예약해 두었습니다.(0: <PAD>, 1: <START>, 2: <OOV>, 3: <UNUSED>) 그래서 실제 상위 빈도 단어 9996개가 사용됩니다.

- 각 단어를 32차원 벡터로 임베딩

실습 - LSTM으로 영화평 분류하기 (imdb 데이터셋 구조)

1장. 자연어처리 인공지능망



3절. Seq2Seq

1장. 자연어처리 인공지능망

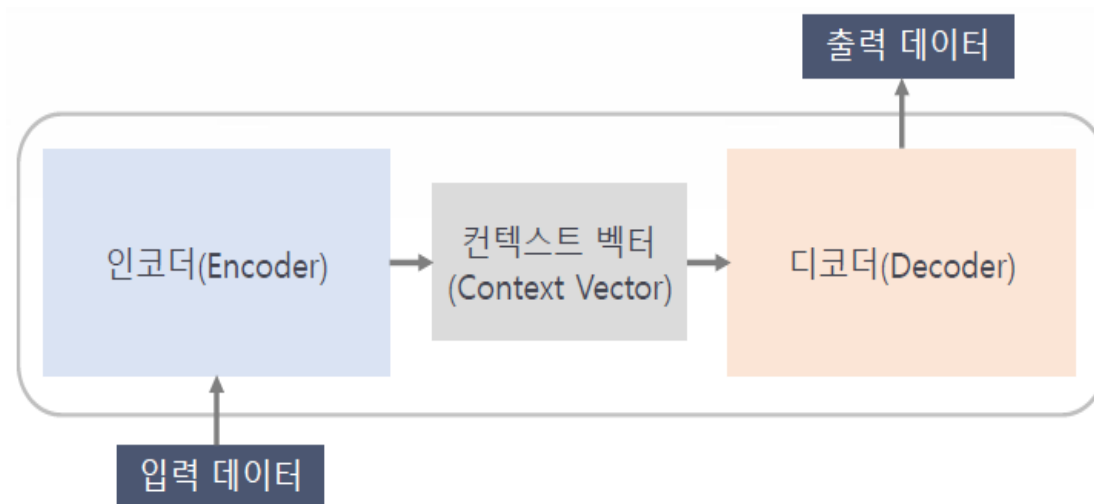


Seq2Seq

1장. 자연어처리 인공지능 / 3절. Seq2Seq

GNMT(Google Neural Machine Translation)

- ▶ 인코더 입력, 디코더 입력, 그리고 디코더 출력으로 나뉨

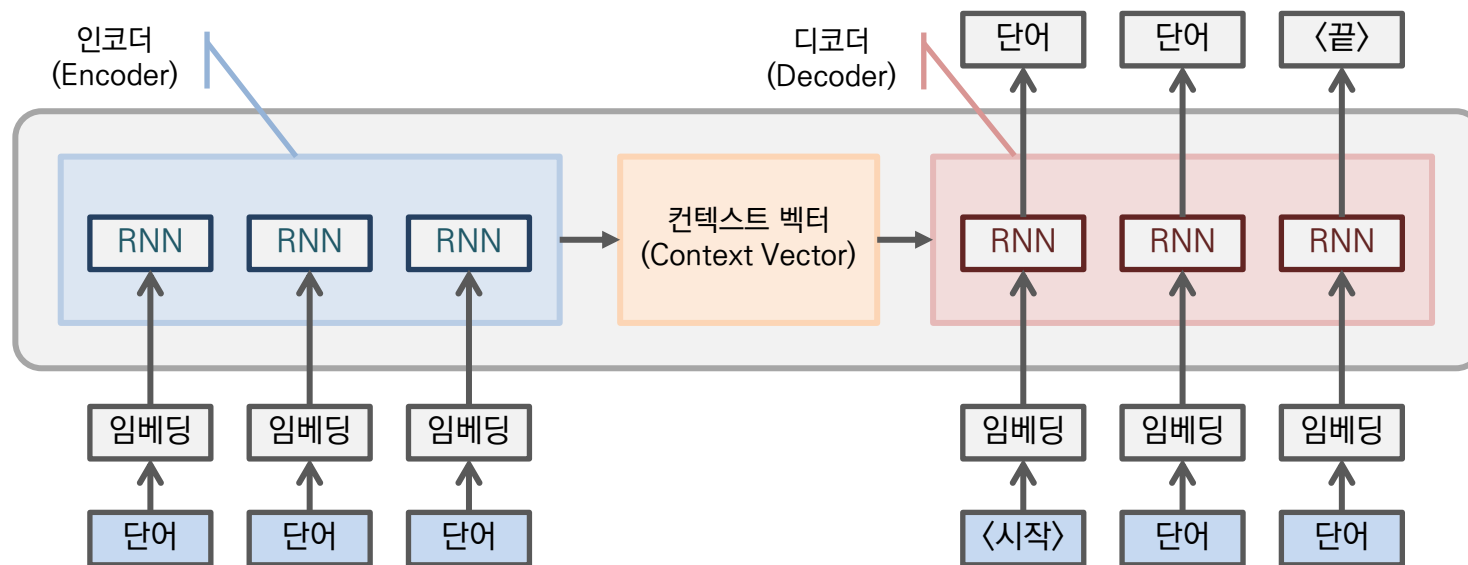


Seq2Seq

1장. 자연어처리 인공지능망 / 3절. Seq2Seq

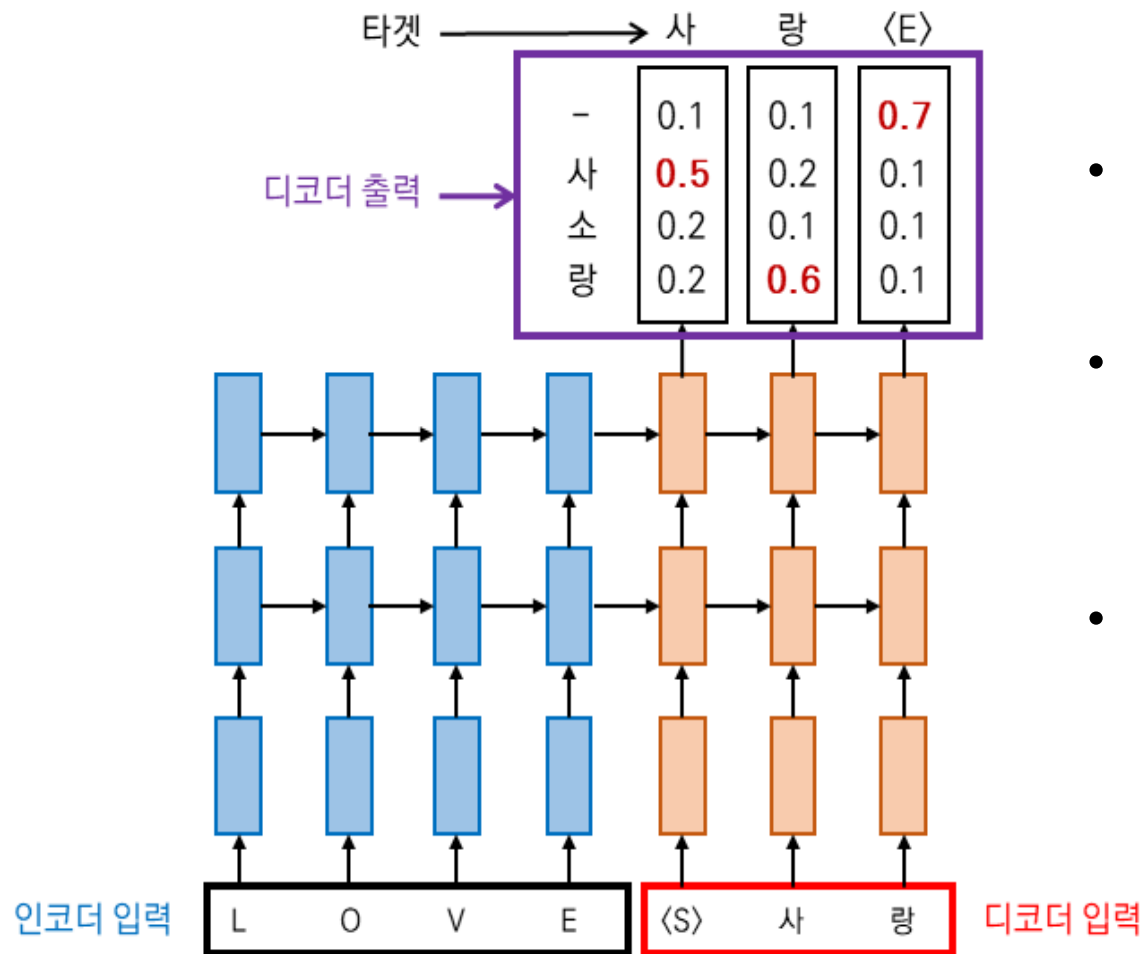
RNN을 연결하여 하나의 시퀀스 데이터를 다른 시퀀스 데이터로 변환하는 모델

- ▶ 시퀀스(Sequence)는 연관된 일련의 데이터를 의미
- ▶ 자연어 문장이 시퀀스에 해당
- ▶ 입력을 처리하기 위한 인코더(Encoder)와 출력을 위한 디코더(Decoder)가 연결되는 구조



GNMT

1장. 자연어처리 인공지능 / 3절. Seq2Seq

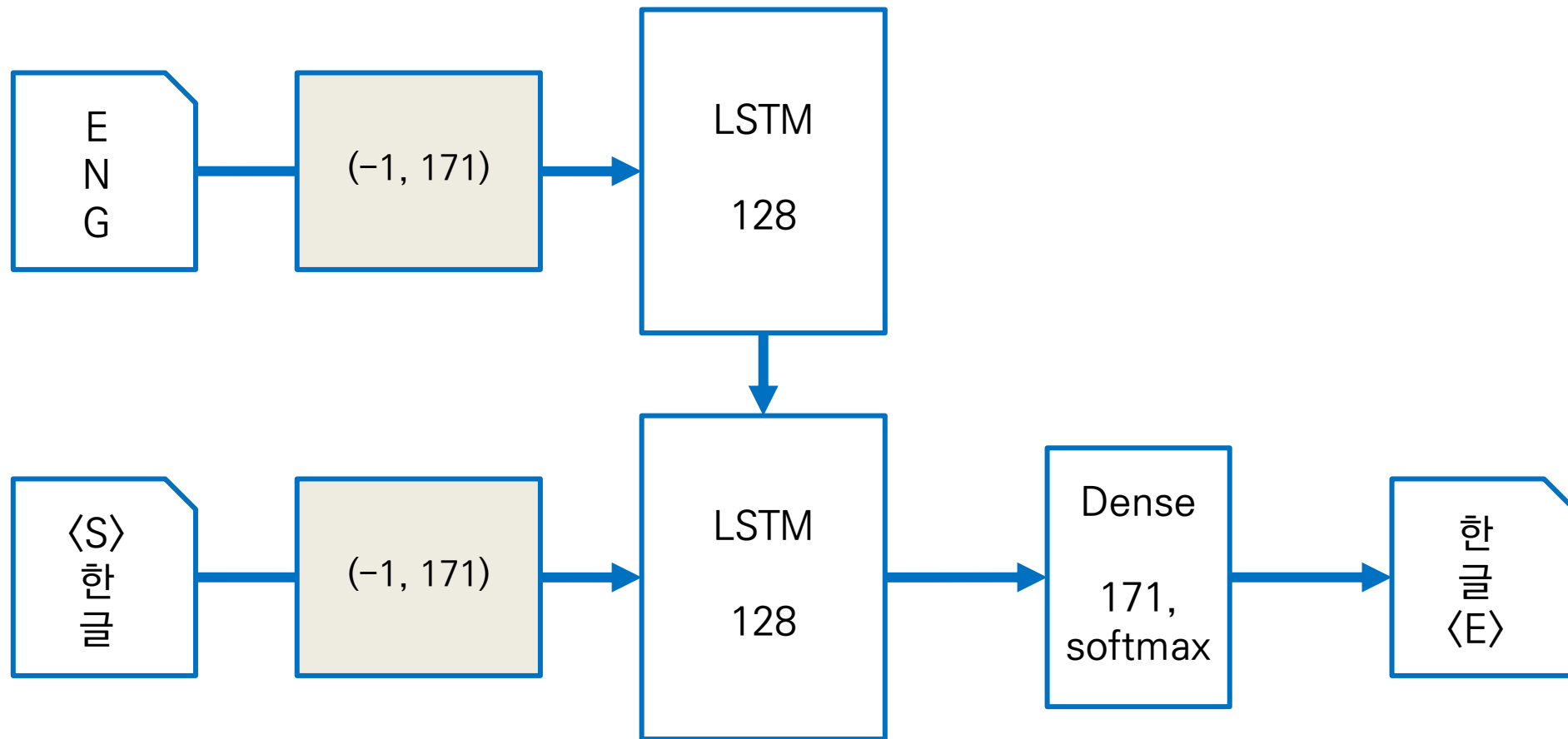


- 인코더는 차례로 입력된 문장들의 모든 단어를 압축하여 컨텍스트 벡터를 생성
- 컨텍스트 벡터는 차대로 입력된 문장의 모든 단어의 정보를 압축한 벡터이며 잠재 벡터(Latent Vector)라고도 함
- 디코더는 입력된 컨텍스트 벡터를 이용하여 출력 시퀀스를 생성하고 출력

예제 - 3장. 4절. 영-한 번역기

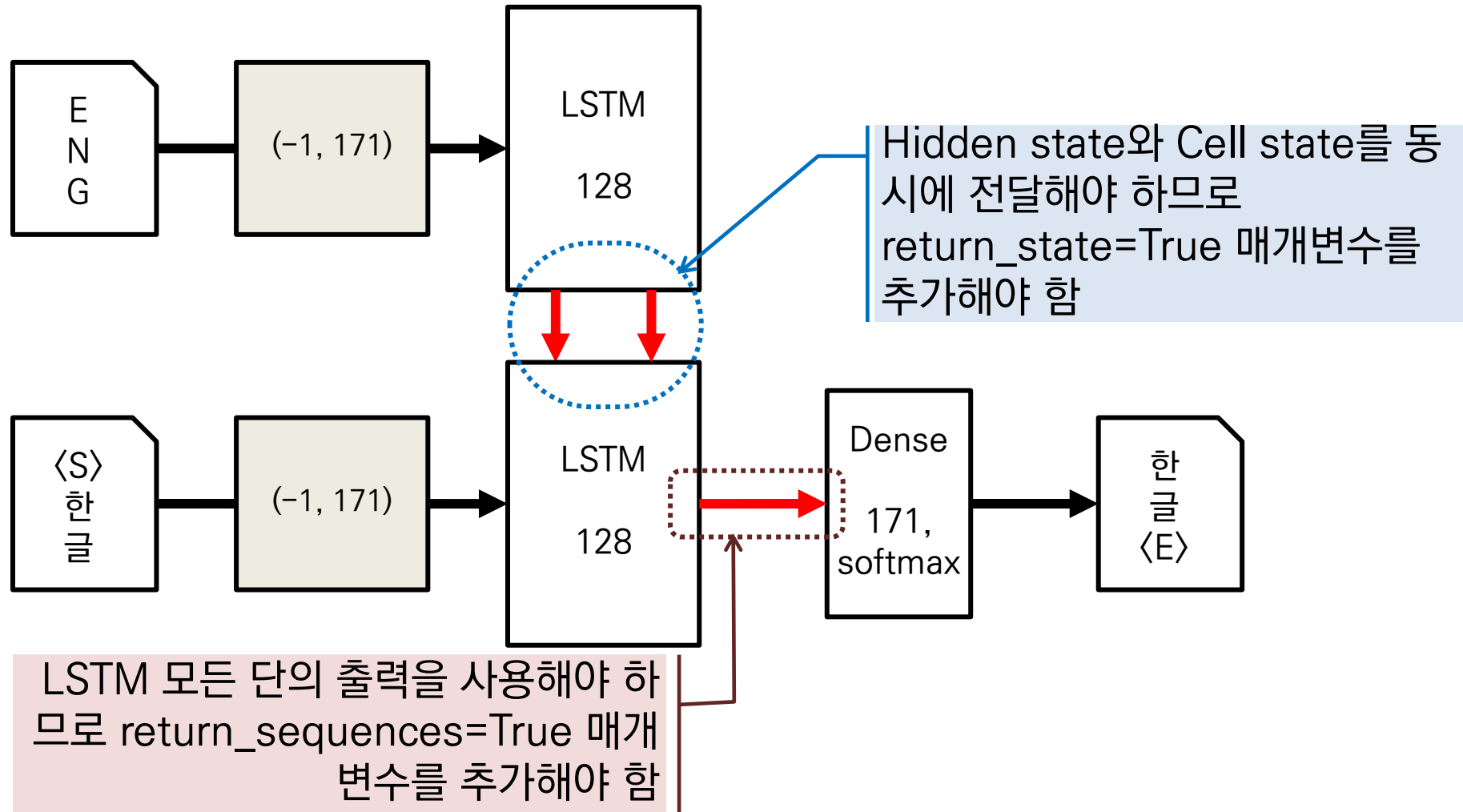
실습 - Seq2Seq를 이용한 영-한 번역기

1장. 자연어처리 인공지능망 / 3절. Seq2Seq



실습 - Seq2Seq를 이용한 영-한 번역기 (Seq2Seq와 LSTM)

1장. 자연어처리 인공지능망 / 3절. Seq2Seq



실습 - Seq2Seq를 이용한 영-한 번역기 (1. 모델 구축)

1장. 자연어처리 인공지능망 / 3절. Seq2Seq

```
1 from tensorflow.keras import Model, layers
2
3 # 입력 단어는 4글자 영단어, 전체문자수는 171개(SEP 포함)
4 enc_input = layers.Input(shape=(4, 171))
5 _, state_h, state_c = layers.LSTM(128, return_state=True)(enc_input)
6
7 dec_input = layers.Input(shape=(3, 171)) # 한글 단어 2자와 start 토큰 1개
8 lstm_out = layers.LSTM(128, return_sequences=True)(dec_input, initial_state=[state_h, state_c])
9
10 dec_output = layers.Dense(171, activation='softmax')(lstm_out)
11
12 model = Model(inputs=[enc_input, dec_input], outputs=dec_output)
13 model.summary()
```

1. 모델 구축
2. 입력값 토큰화, 임베딩
3. 훈련을 정의하고 학습
4. 평가
5. 예측하기 위한 데이터를 토큰화, 임베딩후 predict
6. 라벨 출력

```
1 model.compile(loss='sparse_categorical_crossentropy',
2               optimizer='adam') # metrics가 없으면 훈련시 loss만 출력됨
```

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 4, 171)	0	-
input_layer_1 (InputLayer)	(None, 3, 171)	0	-
lstm (LSTM)	[(None, 128), (None, 128), (None, 128)]	153,600	input_layer[0][0]
lstm_1 (LSTM)	(None, 3, 128)	153,600	input_layer_1[0][0], lstm[0][1], lstm[0][2]
dense (Dense)	(None, 3, 171)	22,059	lstm_1[0][0]

Total params: 329,259 (1.26 MB)
Trainable params: 329,259 (1.26 MB)
Non-trainable params: 0 (0.00 B)

실습 - Seq2Seq를 이용한 영-한 번역기 (2. 입력값 토큰화, 임베딩)

1장. 자연어처리 인공지능 / 3절. Seq2Seq

```
1 # 문자배열 생성
2 import pandas as pd
3 import numpy as np
4 # <START>, <END>, <PAD>
5 arr1 = [c for c in 'SEPabcdefghijklmnopqrstuvwxyz']
6 arr2 = pd.read_csv('korean.csv', header=None)
7 # print(arr2[0].values.tolist())
8 num_to_char = arr1 + arr2[0].values.tolist()
9 print(len(num_to_char))
10 char_to_num = {char:i for i, char in enumerate(num_to_char)}
11 # print(char_to_num)
```

171

```
1 # 학습용 단어셋 불러오기
2 raw = pd.read_csv('translate.csv', header=None)
3 eng_kor = raw.values.tolist()
4 print(len(eng_kor)) # 학습할 전체 단어 수 110개
```

110

```
1 # 단어를 숫자 배열로 변환
2 temp_eng = 'love'
3 temp_eng_n = [char_to_num[c] for c in temp_eng]
4 print(temp_eng_n)
5 temp_kor = '사랑'
6 np.eye(171)[temp_eng_n].shape
```

[14, 17, 24, 7]
(4, 171)

실습 - Seq2Seq를 이용한 영-한 번역기 (3. 학습)

1장. 자연어처리 인공지능 / 3절. Seq2Seq

```
1 # 단어를 원-한 인코딩된 배열로 변환
2 def encode(eng_kor):
3     enc_in = []
4     dec_in = []
5     rnn_out = [] # decoder output
6     for seq in eng_kor:
7         eng = [char_to_num[c] for c in seq[0]]
8         enc_in.append(np.eye(171)[eng])
9
10        kor = [char_to_num[c] for c in ('S'+seq[1])]
11        dec_in.append(np.eye(171)[kor])
12
13        target = [char_to_num[c] for c in (seq[1] + 'E')]
14        rnn_out.append(target)
15
16    enc_in = np.array(enc_in)
17    dec_in = np.array(dec_in)
18    rnn_out = np.array(rnn_out)
19    rnn_out = np.expand_dims(rnn_out, axis=2)
20    return enc_in, dec_in, rnn_out
```

```
1 X_enc, X_dec, y_rnn = encode(eng_kor)
```

```
1 %%time
2 model.fit([X_enc, X_dec], y_rnn, epochs=500)
```

Epoch 1/500

4/4 _____ 3s 11ms/step - loss: 5.1339

Epoch 2/500

4/4 _____ 0s 8ms/step - loss: 5.0933

1. 모델 구축
2. 입력값 토큰화, 임베딩
3. 훈련을 정의하고 학습
4. 평가
5. 예측하기 위한 데이터를 토큰화, 임베딩후 predict
6. 라벨 출력

실습 - Seq2Seq를 이용한 영-한 번역기 (4. 예측)

1장. 자연어처리 인공지능 / 3절. Seq2Seq

```
1 enc_in, dec_in, _ = encode([[ 'tail', 'PP' ]])
2 # print(enc_in)
3 pred = model.predict([enc_in, dec_in])
4 # print(pred.shape)
5 word = np.argmax(pred[0], axis=-1)
6 # print(word)
7 num_to_char[word[0]], num_to_char[word[1]]
```

1/1 ————— 0s 246ms/step
('크', '다')

```
1 from numpy.random import randint
2 pick = randint(0, len(eng_kor), 5)
3 choose = [ [eng_kor[i][0], 'PP'] for i in pick]
4 print(choose)
```

[['thin', 'PP'], ['wing', 'PP'], ['ring', 'PP'], ['goal', 'PP'], ['pick',

```
1 enc_in, dec_in, _ = encode(choose)
2 pred = model.predict([enc_in, dec_in])
3 print(pred.shape)
```

1/1 ————— 0s 251ms/step
(5, 3, 171)

```
1 for i in range(len(choose)):
2     eng = choose[i][0]
3     word = np.argmax(pred[i], axis=-1)
4     kor = ''
5     for j in range(2):
6         kor = kor + num_to_char[word[j]]
7     print(eng, kor)
```

thin 얇은
wing 날개
ring 반지
goal 목적
pick 선택

4절. 어텐션

1장. 자연어처리 인공지능경망

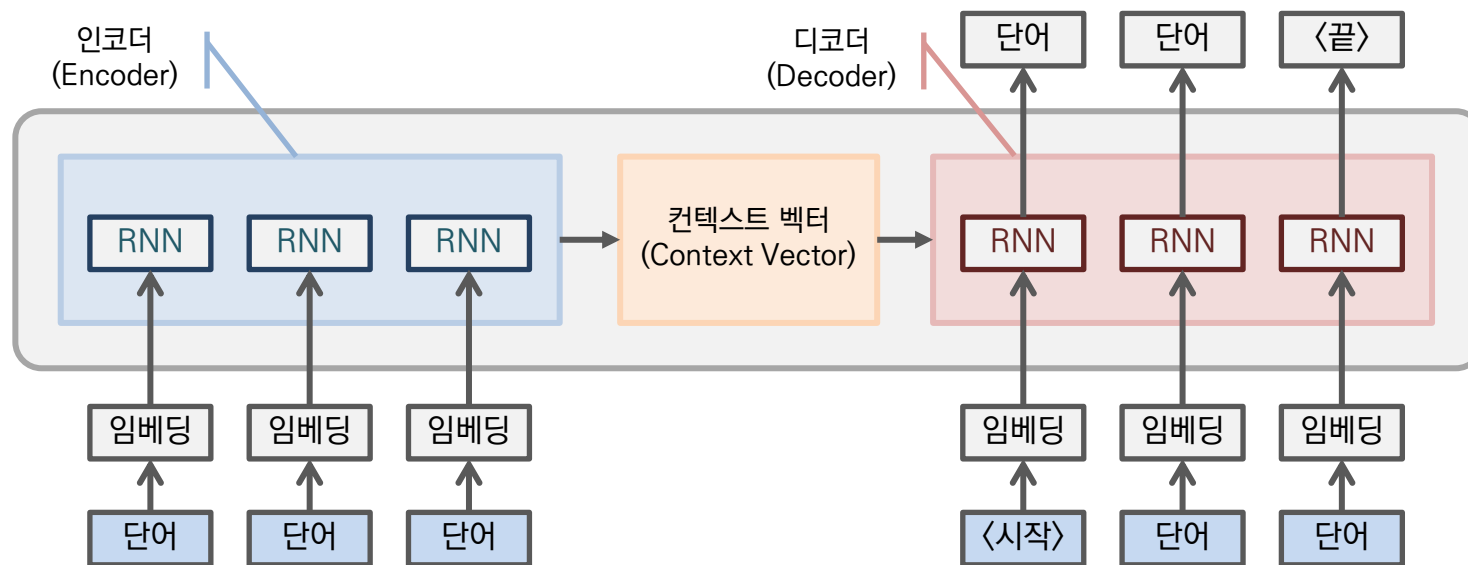


Seq2Seq의 문제점

1장. 자연어처리 인공지능 / 4절. 어텐션

인코더가 입력 시퀀스를 하나의 벡터로 압축하는 과정에서 입력 시퀀스의 일부 정보 손실

- ▶ RNN 구조의 근본적인 문제점
- ▶ 경사 소실(Vanishing Gradient)이 발생할 가능성이 있음
- ▶ 입력 데이터의 길이가 길어지면 성능이 저하되는 현상이 발생할 가능성이 있음



어텐션

1장. 자연어처리 인공지능 / 4절. 어텐션

어텐션 함수는 Q(Query), K(Keys), V(Values)를 매개변수로 사용

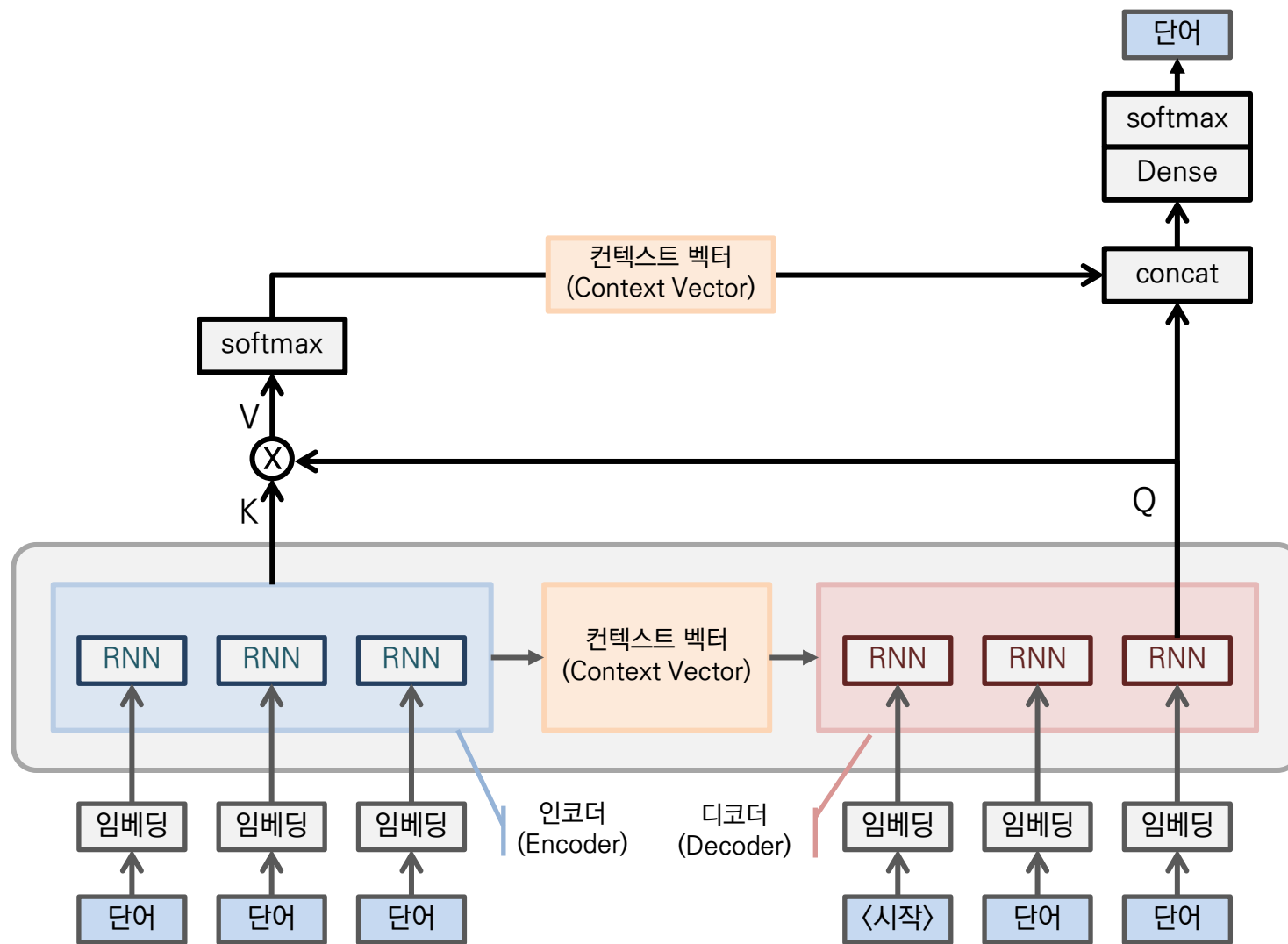
- ▶ Q(Query)는 특정 시점에서의 디코더 셀의 은닉 상태
- ▶ K(Key)는 모든 시점에서의 인코더 셀의 Q를 반영하기 전 은닉 상태
- ▶ V(Value)는 모든 시점에서의 인코더 셀의 Q 반영 후 은닉 상태

$$Attention\ Value = Attention(Q, K, V)$$

- 어텐션 함수는 주어진 질의(Query)에 대해서 모든 키(Key)와 각각의 유사도를 계산
- 계산된 유사도를 키와 매핑되어 있는 각각의 값(Value)에 반영
- 유사도가 반영된 값(Value)을 모두 어텐션 값(Attention Value)으로 반환

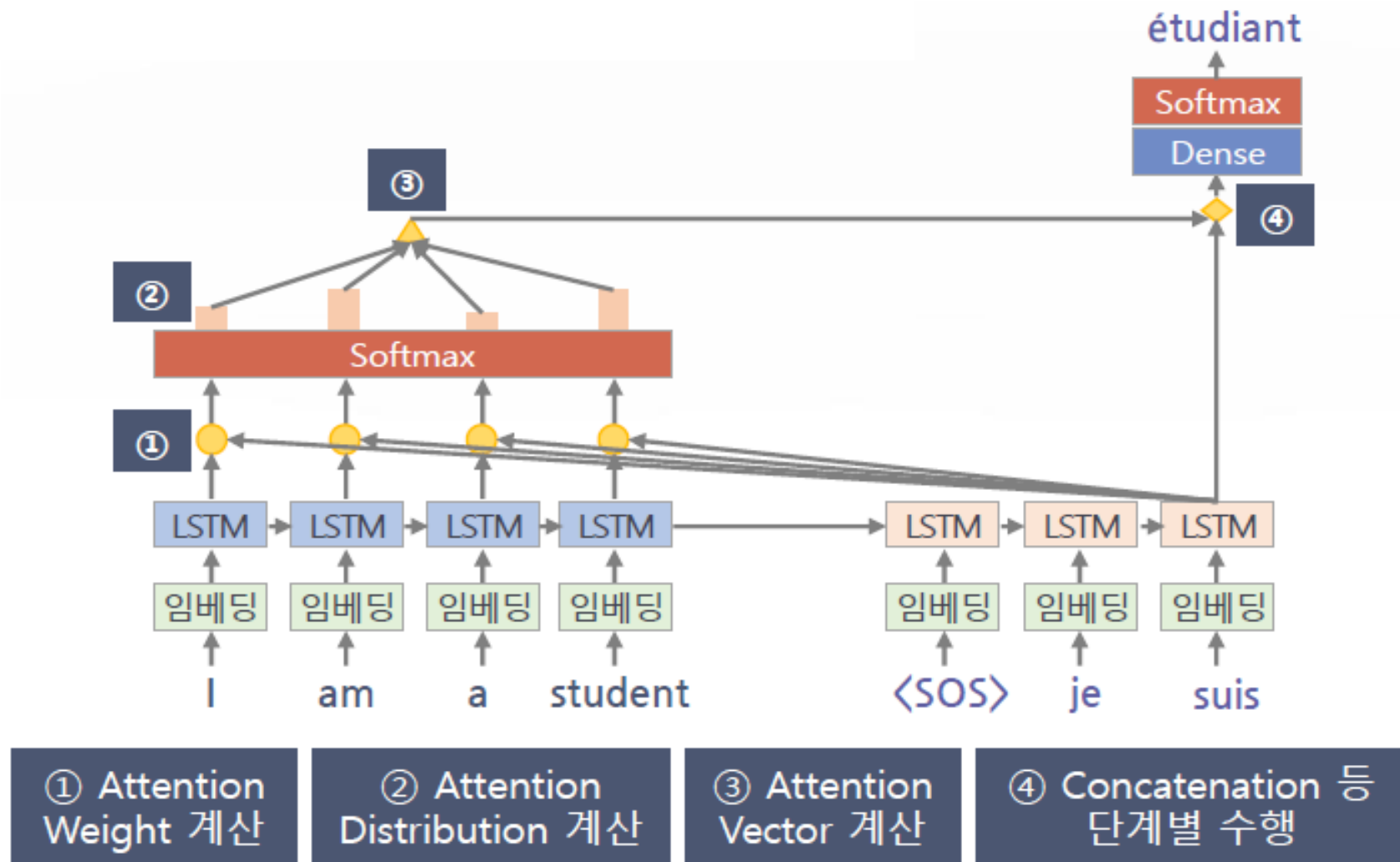
어텐션

1장. 자연어처리 인공지능망 / 4절. 어텐션



어텐션

1장. 자연어처리 인공지능 / 4절. 어텐션



실습 - 어텐션을 이용한 영-한 번역기

2장. 자연어처리 인공지능망 / 4절. 어텐션

```
1 import tensorflow as tf
2 from tensorflow.keras import Model, layers
3
4 # 인코더 입력 정의: 영문 단어가 입력, 각 단어는 4자 길이, 모든 문자의 수는 171개
5 enc_input = layers.Input(shape=(4, 171))
6
7 # 인코더 LSTM 정의: 모든 타임스텝의 출력을 반환하도록 설정
8 enc_output, state_h, state_c = layers.LSTM(128, return_sequences=True, return_state=True)(enc_input)
9
10 # 디코더 입력 정의
11 dec_input = layers.Input(shape=(3, 171)) # 한글 단어는 2자 길이 + <Start> 토큰, 모든 문자의 수는 171개
12
13 # 디코더 LSTM 정의: 모든 시퀀스의 출력을 반환하도록 설정
14 dec_lstm_output, _, _ = layers.LSTM(128, return_sequences=True, return_state=True)(dec_input, initial_state=[state_h, state_c])
15
16 # 어텐션 메커니즘 정의
17 context_vector = layers.Attention()([dec_lstm_output, enc_output])
18
19 # 컨텍스트 벡터와 디코더 LSTM 출력을 결합
20 context_and_lstm_output = layers.Concatenate()([context_vector, dec_lstm_output])
21
22 # 디코더 출력층 정의: 출력 크기는 모든 문자의 수인 171, softmax 활성화 함수를 사용
23 output = layers.Dense(171, activation='softmax')(context_and_lstm_output)
24
25 # 모델 정의: 인코더 입력(enc_input)과 디코더 입력(dec_input)을 모델의 입력으로, 디코더 출력을 모델
26 model = Model(inputs=[enc_input, dec_input], outputs=[output])
27
28 # 모델 요약 출력
29 model.summary()
```

- Seq2Seq 예제 코드의 모델을 수정하세요.
- 나머지 코드는 같습니다.

Model: "functional"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 4, 171)	0	-
input_layer_1 (InputLayer)	(None, 3, 171)	0	-
lstm (LSTM)	[(None, 4, 128), (None, 128), (None, 128)]	153,600	input_layer[0][0]
lstm_1 (LSTM)	[(None, 3, 128), (None, 128), (None, 128)]	153,600	input_layer_1[0][0], lstm[0][1], lstm[0][2]
attention (Attention)	(None, 3, 128)	0	lstm_1[0][0], lstm[0][0]
concatenate (Concatenate)	(None, 3, 256)	0	attention[0][0], lstm_1[0][0]
dense (Dense)	(None, 3, 171)	43,947	concatenate[0][0]

Total params: 351,147 (1.34 MB)
Trainable params: 351,147 (1.34 MB)
Non-trainable params: 0 (0.00 B)

2장. 허깅페이스와 트랜스포머 알아보기

허깅페이스 트랜스포머를 이용한 자연어처리

1절. 트랜스포머

2장. 허깅페이스와 트랜스포머 알아보기

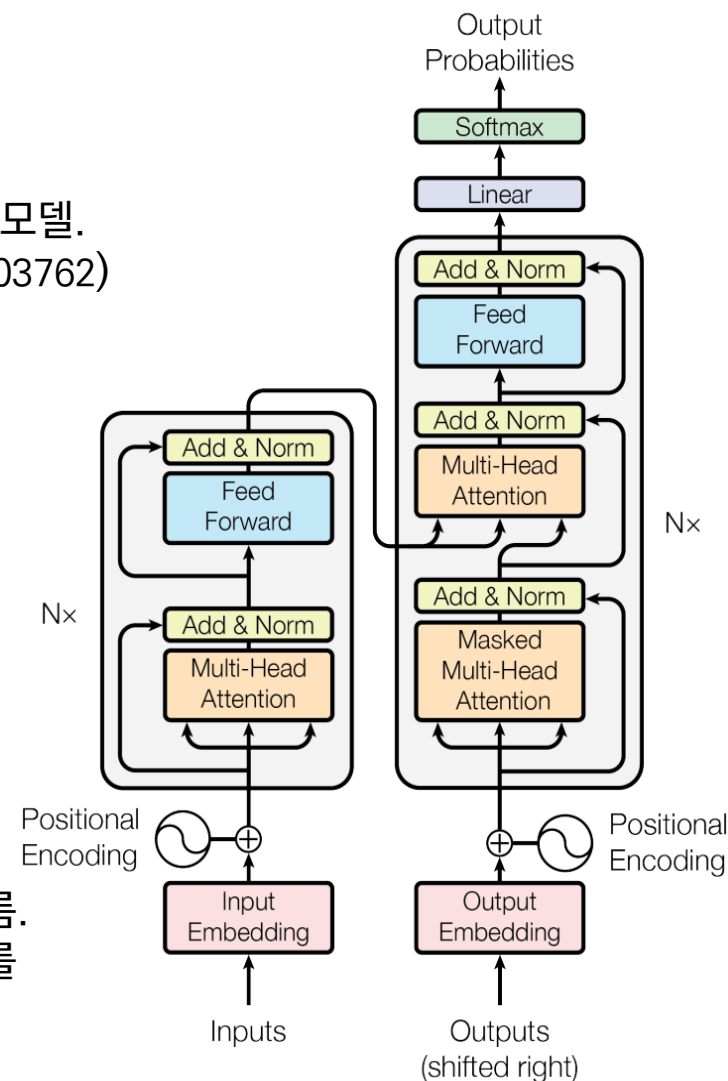


Transformer

2장. 허깅페이스와 트랜스포머 알아보기 / 2절. 허깅페이스의 트랜스포머

✓ 작동 원리

- ▶ GPT 모델은 'Transformer(트랜스포머)'라는 아키텍처를 기반으로 함
 - 트랜스포머 모델은 자연어 처리와 같은 순차적인 데이터를 처리하기 위해 설계된 딥러닝 모델.
 - 2017년 구글이 “Attention is all you need” 논문에서 발표한 모델(<https://arxiv.org/abs/1706.03762>)
- ▶ **입력 처리**: 문장을 단어 단위로 쪼개어 각 단어를 수치로 변환(임베딩)합니다. 이 변환된 숫자 벡터는 모델이 이해할 수 있는 형태로, 각 단어의 의미를 담고 있음.
- ▶ **어텐션 메커니즘 (Attention Mechanism)**: 트랜스포머의 핵심 아이디어는 어텐션 메커니즘입니다. 어텐션은 입력의 모든 단어가 서로 얼마나 중요한지를 계산하여, 중요한 단어에 더 집중할 수 있도록 합니다. 쉽게 말해, 문장 내에서 어떤 단어들이 서로 관련이 있는지를 판단하고, 그 연관성을 바탕으로 정보의 흐름을 조절함.
- ▶ **인코더와 디코더**: 트랜스포머는 인코더와 디코더로 구성.
 - **인코더**: 입력 문장을 받아 어텐션을 통해 의미 있는 벡터로 변환.
 - **디코더**: 인코더에서 얻은 벡터를 받아 새로운 문장을 생성하거나, 번역 작업 등을 수행.
- ▶ **병렬 처리**: 트랜스포머는 RNN이나 LSTM과 달리 모든 단어를 한 번에 처리할 수 있어 학습이 빠름. 이는 어텐션 메커니즘 덕분에 가능하며, 이전 단어들을 순차적으로 보지 않아도 각 단어의 중요도를 빠르게 계산할 수 있음.



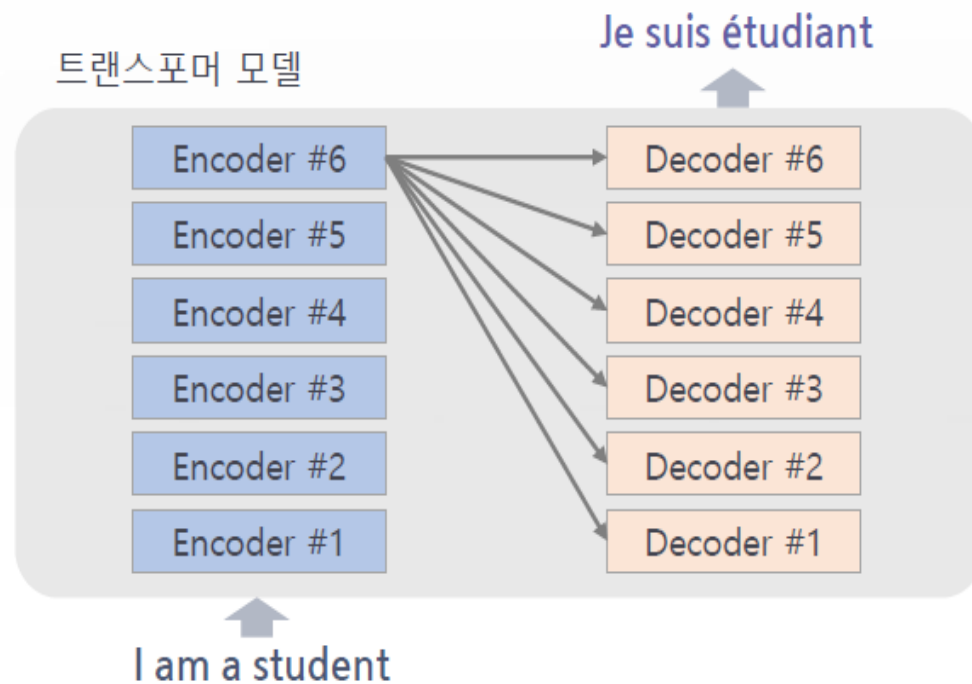
트랜스포머의 인코더와 디코더 구조

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

인코딩 컴포넌트는 여러 인코더로 구성되어 있음

디코딩 컴포넌트는 인코딩 컴포넌트와 같은 개수의 디코더로 구성

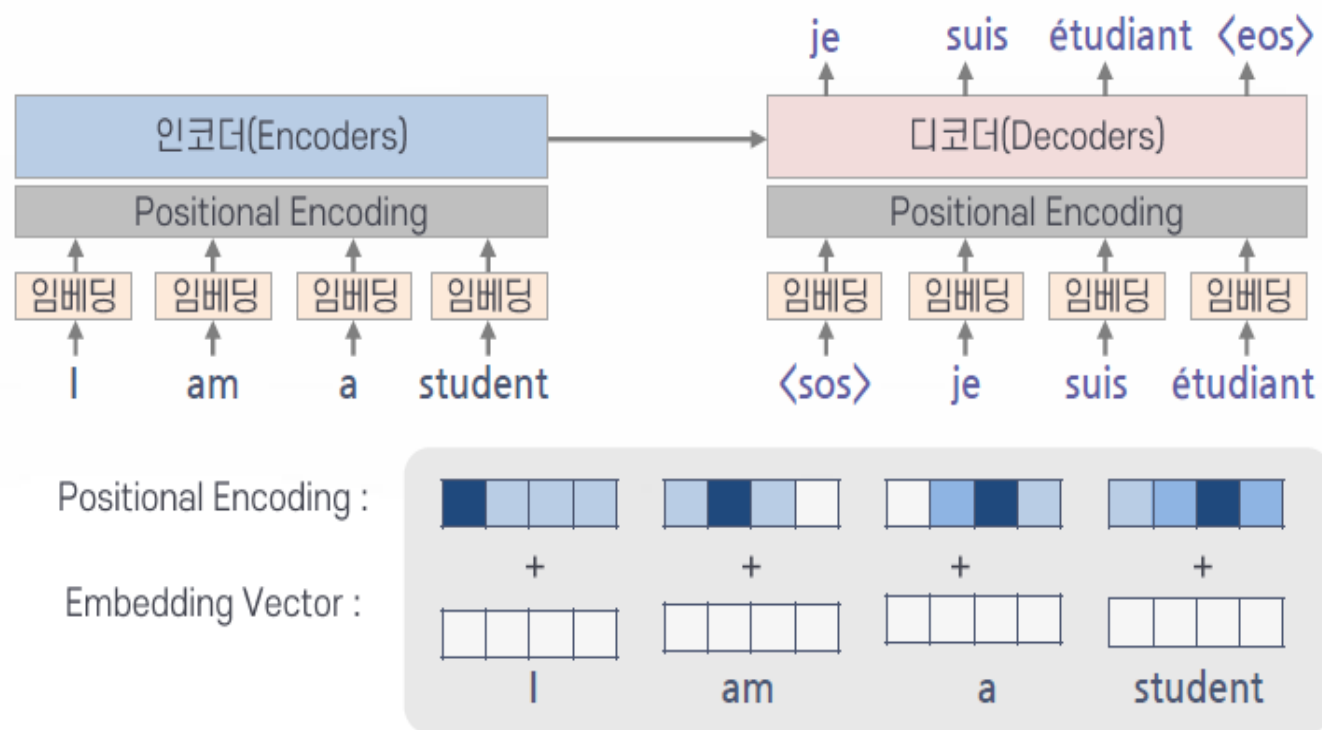
- ▶ 논문에는 6개의 인코더와 디코더로 구성되어 있으나 임의의 개수로 변경 가능



트랜스포머의 임베딩과 Positional 인코딩

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

Positional 인코딩은 트랜스포머에서 각 입력 단어의 위치 정보를 부여하기 위해 각 단어의 임베딩 벡터에 위치 정보들을 추가하여 모델의 입력으로 사용하는 방법



트랜스포머의 Self-Attention

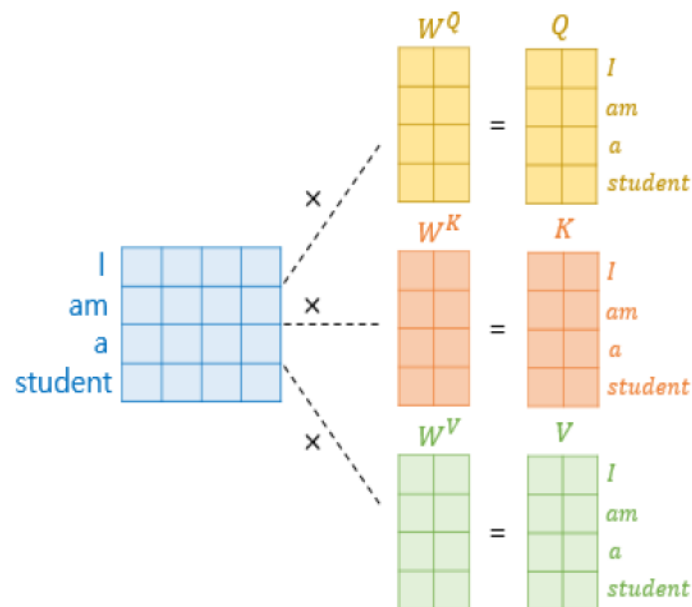
2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

트랜스포머의 Self-Attention은 현재 처리 중인 단어에 대해 다른 연관 있는 단어들과의 맥락을 파악하는 방법을 제공

‘The animal didn’t cross the street because it was too tired’

이 문장의 경우 “it”이 가리키는 것은 무엇일까요?
animal일까요? 아니면 street일까요?

- Self-Attention의 첫 단계는 입력 문장에 대해 Query, Key, Value를 계산하는 것
- 입력 문장의 512 크기의 벡터와 학습할 가중치(W^Q , W^K , W^V)를 곱하여 64 크기의 Query, Key, Value 벡터를 생성



트랜스포머의 Self-Attention

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

- Query에 Key의 전치(Transpose) 행렬을 곱해서 내적을 계산
- 만일 Query와 Key가 특정 문장에서 중요한 역할을 하고 있다면 트랜스포머는 이들 사이의 내적(Dot Product)값을 크게 하는 방향으로 학습
- 내적 값이 커지면 해당 Query와 Key가 벡터 공간상 가까이 있을 확률이 높음
- Key의 벡터 크기인 64의 제곱근인 8로 나눈 후 소프트맥스 함수를 적용
- Value와 내적을 곱하여 어텐션 값인 Z를 계산

Diagram illustrating the Self-Attention mechanism:

Query matrix Q (yellow) is multiplied by the transpose of the Key matrix K^T (orange). The result is passed through a softmax function, which also divides by the square root of the key dimension $\sqrt{d_k}$. This result is then multiplied by the Value matrix V (green) to produce the Attention Value Matrix a (blue).

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

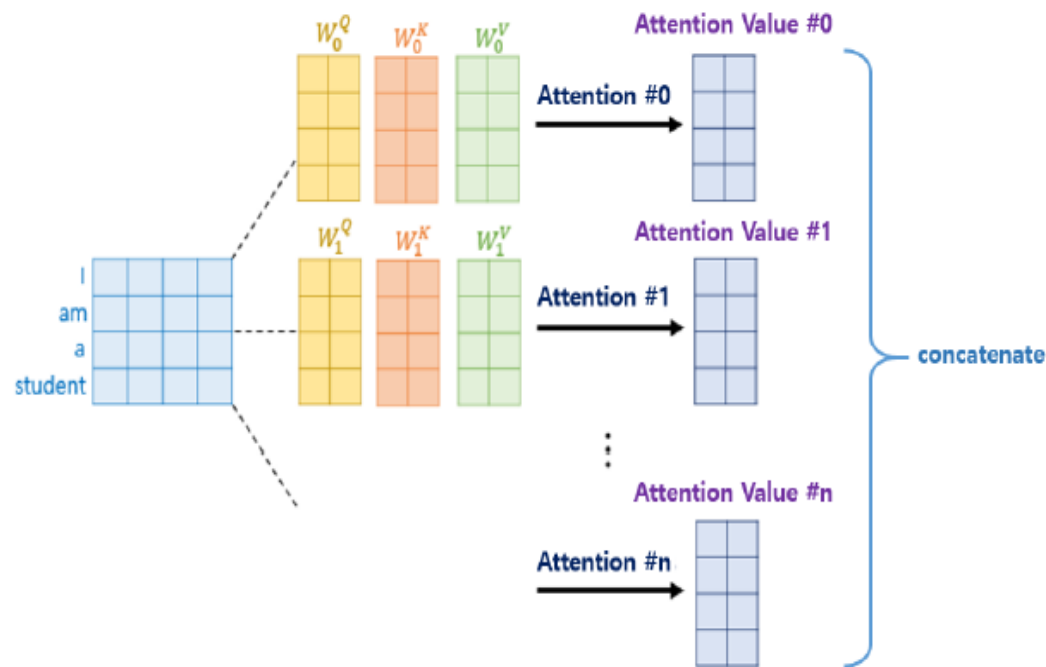
위 수식에서 $\sqrt{d_k}$ 로 나눈 이유는 Query와 Key의 내적 행렬의 분산(Variance)을 축소하고 경사 소실(Gradient Vanishing)을 발생을 방지하기 위해서입니다.

Multi-head Self Attention

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

Attention을 병렬로 수행함으로써 다양한 관점에서 단어 간 관계 정보 파악이 가능

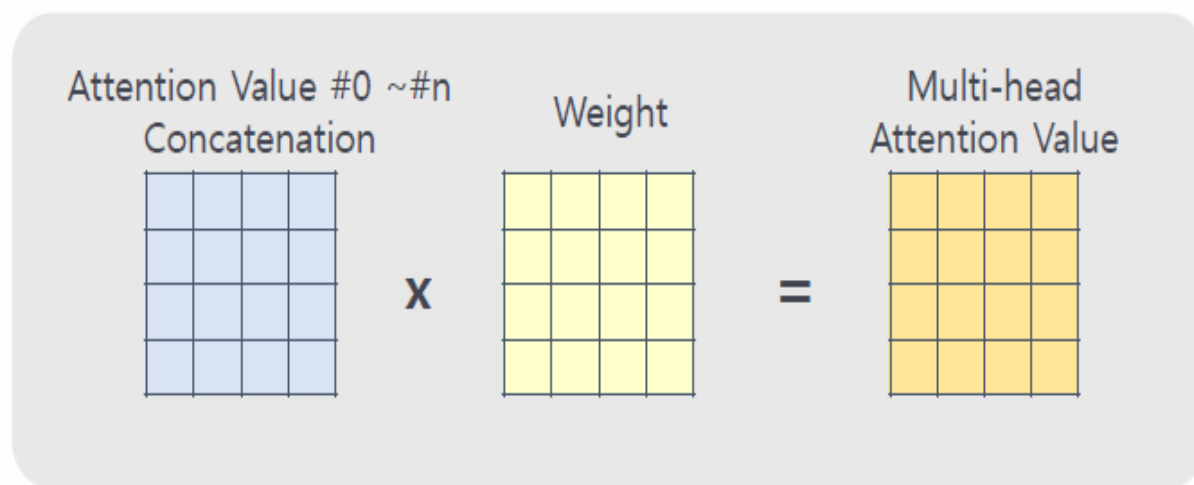
Input 문장에 대해 N개의 Query, Key, Value를 계산하기 위한 N개의 가중치(W^Q , W^K , W^V)를 생성하여 병렬로 Self Attention 수행하고 N개의 Attention Value를 계산



Multi-head Attention Value Matrix

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

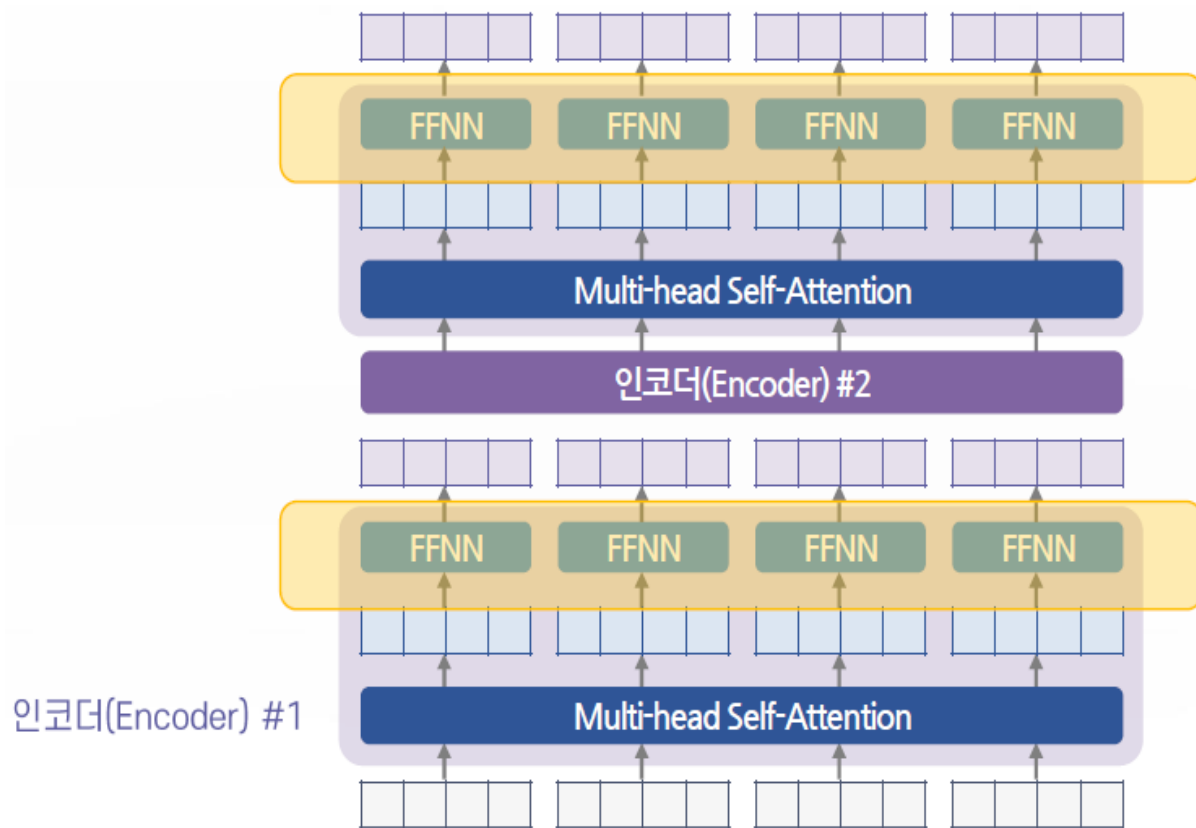
Multi-head Attention 수행 결과를 연결하고 학습할 가중치를 곱하여 Multi-head Attention Value Matrix를 최종 결과로 도출



Position-wise FFN

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

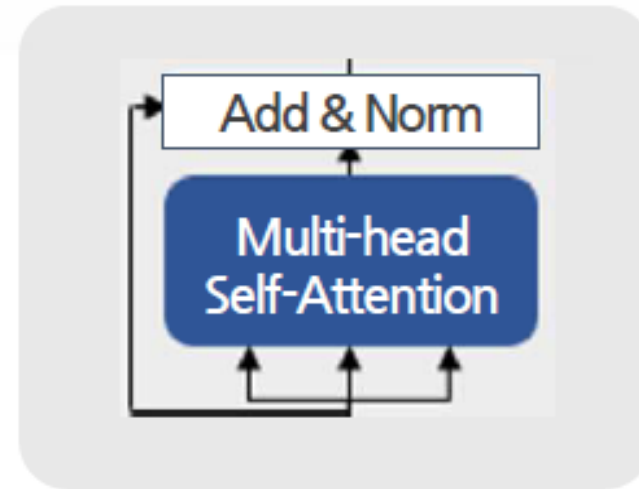
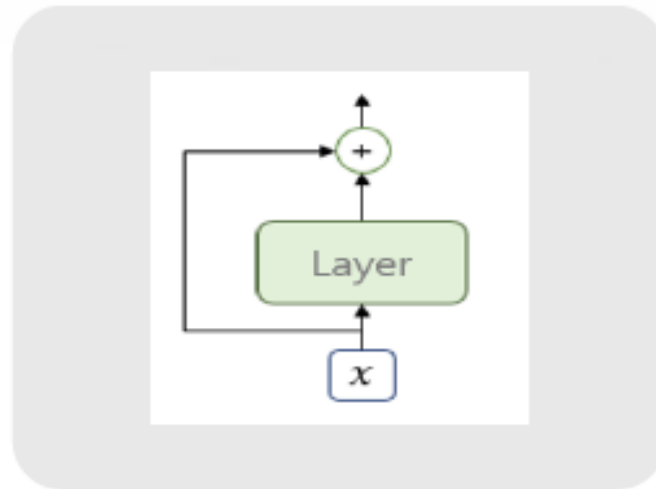
Position-wise FFN은 인코더와 디코더에서 공통으로 포함된 Sub-layer



잔차 연결


2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

경사가 줄어드는 문제를 해결하기 위해 구글은 ResNet을 통해 잔차연결을 적용



정규화

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머



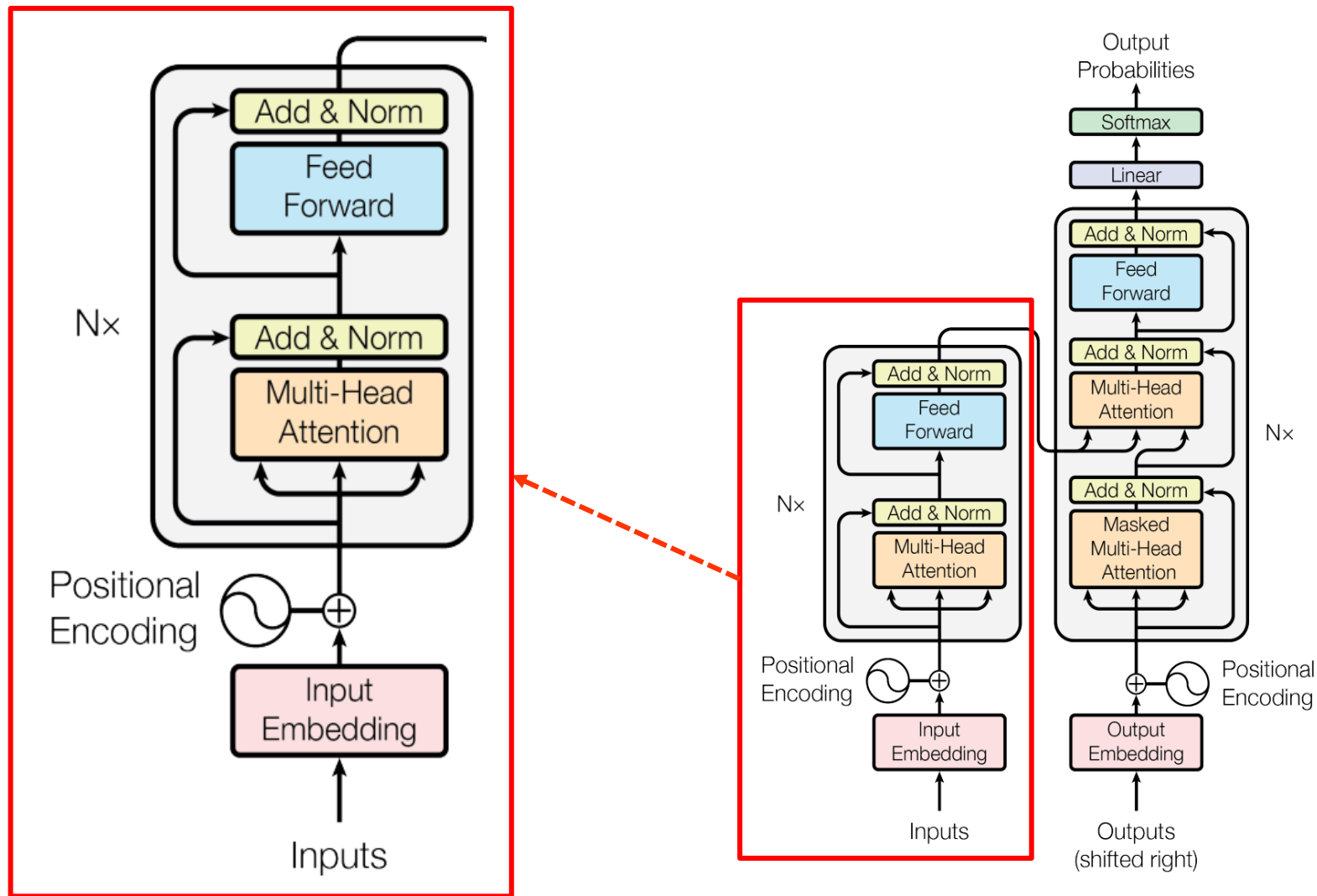
Residual Connection으로 전달된 입력값과 Multi-head Attention Value를 더한 후 정규화(Normalization)를 수행



Position-wise FFNN의 입력값으로 전달하기 전 정규화를 수행함으로써 과적합(Overfitting)을 방지할 수 있음

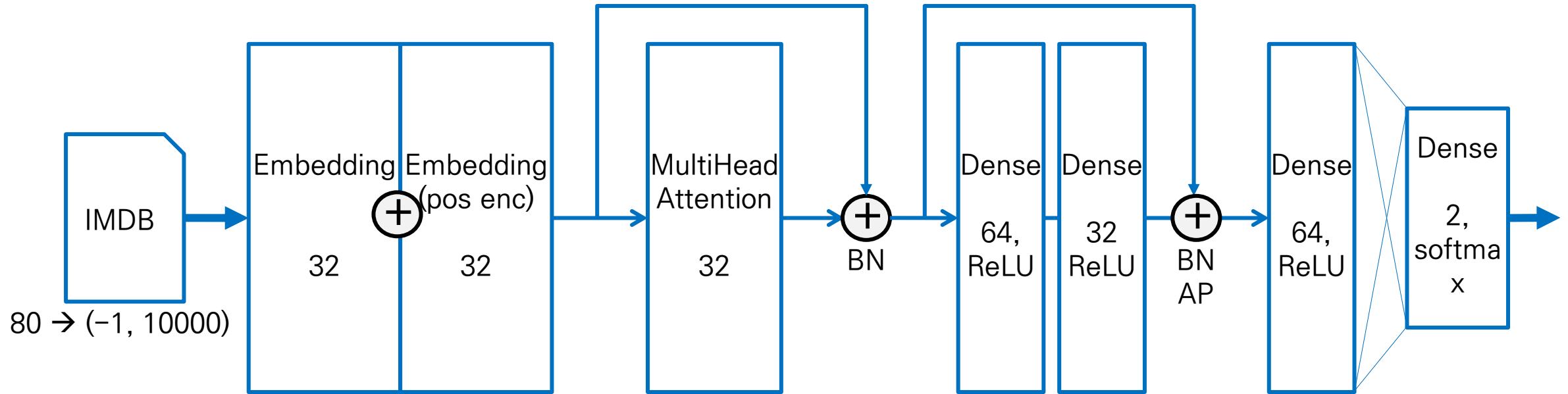
실습 - 트랜스포머의 인코더 블록을 이용한 영화평 분류

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머



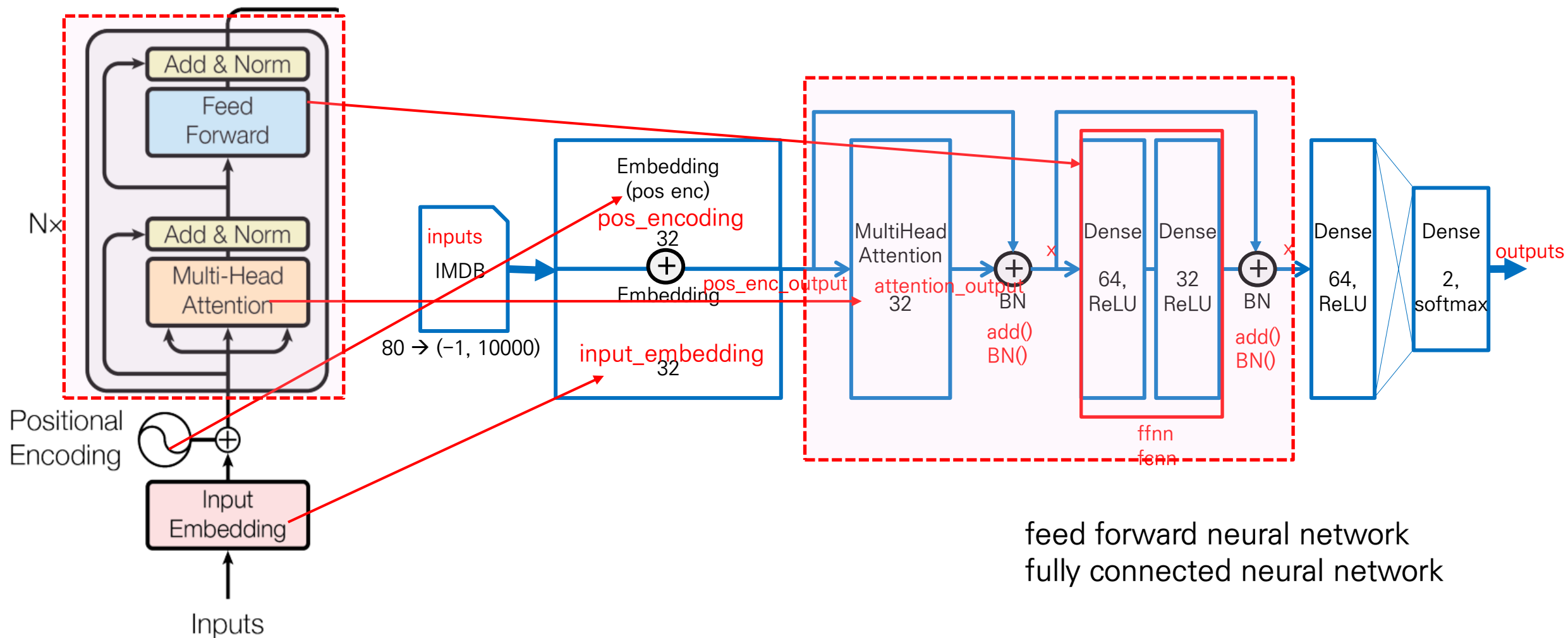
실습 - 트랜스포머의 인코더 블록을 이용한 영화평 분류 (모델 구조)

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머



실습 - 트랜스포머의 인코더 블록을 이용한 영화평 분류 (모델 구조)

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머



실습 - 트랜스포머의 인코더 블록을 이용한 영화평 분류 (모델 구조)

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

```
1 import tensorflow as tf
2 from tensorflow.keras import layers
3 from tensorflow.keras.models import Sequential, Model
4
5 inputs = layers.Input(shape=(80,))
6
7 input_embedding = layers.Embedding(input_dim=10000, output_dim=32)(inputs)
8 positions = tf.range(start=0, limit=80)
9 pos_encoding = layers.Embedding(input_dim=80, output_dim=32)(positions)
10 pos_enc_output = pos_encoding + input_embedding
11
12 attention_output = layers.MultiHeadAttention(num_heads=3, key_dim=32)(pos_enc_output, pos_enc_output)
13 x = layers.add([pos_enc_output, attention_output])
14 x = layers.BatchNormalization()(x)
15
16 ffnn = Sequential([layers.Dense(64, activation="relu"),
17                    layers.Dense(32, activation="relu")])(x)
18 x = layers.add([ffnn, x])
19 x = layers.BatchNormalization()(x)
20 x = layers.GlobalAveragePooling1D()(x)
21 x = layers.Dropout(0.1)(x)
22
23 x = layers.Dense(64, activation="relu")(x)
24 x = layers.Dropout(0.1)(x)
25
26 outputs = layers.Dense(2, activation="softmax")(x)
27 model = Model(inputs=inputs, outputs=outputs)
28
29 model.summary()
```

- 'RNN 실습 - 영화평 분류' 예제 코드에서 모델만 수정하세요.

Model: "functional_1"

Layer (type)	Output Shape	Param #	Connected to
input_layer (InputLayer)	(None, 80)	0	-
embedding (Embedding)	(None, 80, 32)	320,000	input_layer[0][0]
add (Add)	(None, 80, 32)	0	embedding[0][0]
multi_head_attention (MultiHeadAttention)	(None, 80, 32)	12,608	add[0][0], add[0][0]
add_1 (Add)	(None, 80, 32)	0	add[0][0], multi_head_attention[...]
batch_normalization (BatchNormalization)	(None, 80, 32)	128	add_1[0][0]
sequential (Sequential)	(None, 80, 32)	4,192	batch_normalization[0...]
add_2 (Add)	(None, 80, 32)	0	sequential[0][0], batch_normalization[0...]
batch_normalization_1 (BatchNormalization)	(None, 80, 32)	128	add_2[0][0]
global_average_pooling1d (GlobalAveragePooling1D)	(None, 32)	0	batch_normalization_1...
dropout_1 (Dropout)	(None, 32)	0	global_average_poolin...
dense_2 (Dense)	(None, 64)	2,112	dropout_1[0][0]
dropout_2 (Dropout)	(None, 64)	0	dense_2[0][0]
dense_3 (Dense)	(None, 2)	130	dropout_2[0][0]

Total params: 339,298 (1.29 MB)
Trainable params: 339,170 (1.29 MB)
Non-trainable params: 128 (512.00 B)

트랜스포머의 인코더 블록을 이용한 영화평 분류

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

```
[2] 1 model.compile(loss='sparse_categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

```
[3] 1 from tensorflow.keras.datasets import imdb  
2 (X_train, y_train), (X_test, y_test) = imdb.load_data(num_words=10000)  
3 print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

➡ Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/imdb.npz>
17464789/17464789 ----- **0s** 0us/step
(25000,) (25000,) (25000,) (25000,)

```
[4] 1 from tensorflow.keras.preprocessing.sequence import pad_sequences  
2 X_train_pad = pad_sequences(X_train, maxlen=80, truncating='post', padding='post')  
3 X_test_pad = pad_sequences(X_test, maxlen=80, truncating='post', padding='post')
```

▶ 1 %%time
2 model.fit(X_train_pad, y_train, epochs=10, batch_size=200)

... Epoch 1/10
125/125 ----- **40s** 280ms/step - accuracy: 0.6867 - loss: 0.5733

트랜스포머의 인코더 블록을 이용한 영화평 분류

2장. 허깅페이스와 트랜스포머 알아보기 / 1절. 트랜스포머

```
[6] 1 model.evaluate(X_test_pad, y_test)
```

↔ 782/782 ----- 13s 16ms/step - accuracy: 0.7675 - loss: 1.7022
[1.7143645286560059, 0.7689999938011169]

```
[7] 1 print(X_test_pad.shape)
```

↔ (25000, 80)

```
[8] 1 import numpy as np  
2 pred = model.predict(X_test_pad)  
3 # pred = (pred > 0.5).astype(int)  
4 pred = np.argmax(pred, axis=1)
```

↔ 782/782 ----- 11s 14ms/step

```
[9] 1 from sklearn.metrics import confusion_matrix  
2 print(confusion_matrix(y_test, pred))
```

↔
[[10063 2437]
 [3338 9162]]

2절. 허깅페이스의 트랜스포머

2장. 허깅페이스와 트랜스포머 알아보기



허깅페이스(Hugging Face)

2장. 허깅페이스와 트랜스포머 알아보기 / 2절. 허깅페이스의 트랜스포머

✓ 자연어 처리와 관련된 다양한 기술과 리소스를 제공하는 회사 및 온라인 플랫폼

▶ <https://huggingface.co/>

✓ Transformers 라이브러리

- ▶ Transformer 모델들을 쉽게 사용할 수 있도록 도와주는 오픈소스 라이브러리
- ▶ 최신의 딥 러닝 기반 자연어 처리 모델을 쉽게 사용할 수 있도록 지원함
- ▶ BERT, GPT, RoBERTa 등 다양한 사전 훈련된 모델들을 포함하고 있으며, 다양한 NLP 작업에 대해 미세 조정(Fine-tuning)이 가능

• 허깅페이스는 커뮤니티와의 협력을 강조하며, 다양한 NLP 기술 발전에 기여하는 것을 목표로 하고 있습니다. 이들의 라이브러리와 도구들은 NLP 개발자들에게 효율적이고 강력한 도구를 제공하여, 복잡한 자연어 이해와 생성 문제를 해결하는 데 도움을 줍니다.

✓ 사전 훈련된 모델 호스팅

- ▶ 다양한 사전 훈련된 언어 모델을 제공하며, 이를 통해 개발자들은 새로운 NLP 애플리케이션을 빠르게 구축하고 테스트할 수 있음
- ▶ 모델을 사용할 때 필요한 인퍼런스(추론) 환경도 제공함

✓ Hub

- ▶ 허깅페이스 Hub는 커뮤니티가 공유한 다양한 모델, 토큰나이저, 데이터셋 등의 리소스를 검색하고 공유할 수 있는 플랫폼
- ▶ 개발자들은 Hub을 통해 다른 사람들이 공유한 리소스를 활용하여 자신의 프로젝트를 보다 효율적으로 개발할 수 있음

✓ Datasets 라이브러리

- ▶ 다양한 공개 데이터셋을 손쉽게 접근하고 활용할 수 있는 라이브러리인 datasets를 제공
- ▶ 이 라이브러리는 데이터셋을 로드하고 전처리하는 기능을 지원하여, NLP 모델 훈련 및 평가에 유용하게 사용됨

<https://huggingface.co/welcome>

허깅페이스의 Transformers

2장. 허깅페이스와 트랜스포머 알아보기 / 2절. 허깅페이스의 트랜스포머



Transformers

- 자연어 처리(NLP)와 자연어 생성(NLG) 등 다양한 AI 모델을 쉽게 사용할 수 있게 해주는 오픈 소스 라이브러리
- BERT, GPT-3, T5 등 유명한 사전 학습된 트랜스포머 모델을 포함하며, 텍스트 생성, 번역, 감정 분석, 요약 등 다양한 언어 처리 작업을 지원함.



주요 기능 및 특징

- 사전 학습된 모델: 허깅페이스는 BERT, GPT, T5, RoBERTa 등 다양한 사전 학습된 모델을 제공함. 이 모델들은 Hugging Face Model Hub에서 쉽게 다운로드하여 사용할 수 있음.
- 다양한 태스크 지원: 트랜스포머 모델을 통해 감정 분석, 번역, 요약, 질의 응답, 텍스트 생성, 텍스트 분류, 개체명 인식(NER) 등 여러 NLP 작업을 수행할 수 있음.
- 사용자 친화적 인터페이스: Hugging Face 라이브러리는 Python 기반으로 직관적인 API를 제공하며, 초보자부터 전문가까지 쉽게 사용할 수 있도록 설계되었음.
- 모델 미세 조정(Fine-tuning): 사용자는 자신만의 데이터셋으로 사전 학습된 모델을 미세 조정하여 특정 작업에 최적화된 모델을 만들 수 있음
- 허브(Hugging Face Model Hub): 이곳에서 다양한 사전 학습된 모델을 다운로드하거나, 자신이 만든 모델을 업로드하고 공유할 수 있음. 연구자나 개발자가 각자의 모델을 쉽게 공유하고 활용할 수 있는 플랫폼임.
- 프레임워크 호환성: PyTorch와 TensorFlow 두 가지 주요 딥러닝 프레임워크를 모두 지원하며, 동일한 코드로 두 프레임워크에서 실행할 수 있는 높은 유연성을 제공함
- 통합 API: 허깅페이스는 API도 제공하여 로컬에서 모델을 실행하지 않고도 REST API 방식으로 자연어 처리 기능을 활용할 수 있음

영화평 분류 모델

2장. 허깅페이스와 트랜스포머 알아보기 / 2절. 허깅페이스의 트랜스포머

- 허깅페이스(Hugging Face) 트랜스포머 모델 중 영화 평 분류에 가장 널리 사용되는 모델은 BERT와 RoBERTa 기반 모델들임.
- IMDb 데이터셋을 사용한 영화 평 분류에 적합한 모델로는 "distilbert-base-uncased-finetuned-sst-2-english"와 "roberta-base" 모델이 많이 사용됨.
- 이 모델들은 감정 분석(Sentiment Analysis)에 특히 강력한 성능을 보임.



추천 모델

- distilbert-base-uncased-finetuned-sst-2-english
 - SST-2 데이터셋(Stanford Sentiment Treebank)으로 미세 조정된 BERT의 경량화 버전임.
 - 영화 평에서 감정(긍정/부정)을 분류하는 데 뛰어남.
- roberta-base
 - RoBERTa는 BERT의 변형
 - 더 많은 데이터로 학습되어 텍스트 분류 작업에 높은 성능을 보여줌.

영화평 분류 예시

2장. 허깅페이스와 트랜스포머 알아보기 / 2절. 허깅페이스의 트랜스포머

! pip install tf_keras

```
from transformers import pipeline
```

```
# 감정 분석 파이프라인 설정
```

```
sentiment_analysis = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")
```

```
# 예시 영화 평
```

```
reviews = [
```

```
    "This movie was absolutely fantastic! The acting, the plot, everything was perfect.",
```

```
    "I did not enjoy this movie at all. The story was predictable and the acting was terrible.",
```

```
    "A solid performance by the lead actor, but the plot could have been better.",
```

```
    "What a waste of time! I can't believe I sat through the whole movie."
```

```
]
```

```
# 영화 평 분석
```

```
results = sentiment_analysis(reviews)
```

```
# 결과 출력
```

```
for review, result in zip(reviews, results):
```

```
    print(f"Review: {review}\nSentiment: {result['label']} with score {result['score']:.2f}\n")
```

Transformers 인코딩 부분

```
import tensorflow as tf
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential, Model

INPUTS = layers.Input(shape=(MY_LENGTH,)) # 80

INPUT_EMBEDDING = layers.Embedding(input_dim=MY_WORDS, # 10000
                                   output_dim=MY_EMBED)(INPUTS)# 32

# Positional Encoding
POSITIONS = tf.range(start=0,
                     limit=MY_LENGTH)
POS_ENCODING = layers.Embedding(input_dim=MY_LENGTH, output_dim=MY_EMBED)(POSITIONS)
POS_ENC_OUTPUT = POS_ENCODING + INPUT_EMBEDDING

ATTENTION_OUTPUT = layers.MultiHeadAttention(num_heads=3,
                                             key_dim=MY_EMBED)(POS_ENC_OUTPUT,
                                                             POS_ENC_OUTPUT)
X = layers.add([POS_ENC_OUTPUT, ATTENTION_OUTPUT])
X = layers.BatchNormalization()(X)

# FeedForward Network
FFN = Sequential([layers.Dense(MY_HIDDEN, activation="relu"),
                 layers.Dense(MY_EMBED, activation="relu")])
X = layers.add([FFN, X])
X = layers.BatchNormalization()(X)
# 하나의 벡터로 압축해서 Dense로 보내줌
X = layers.GlobalAveragePooling1D()(X)
X = layers.Dropout(0.1)(X)

X = layers.Dense(MY_HIDDEN, activation="relu")(X)
X = layers.Dropout(0.1)(X)

OUTPUTS = layers.Dense(2, activation="softmax")(X)
model = Model(inputs=INPUTS, outputs=OUTPUTS)

model.summary()
```

Transformers 인코딩 부분

```
model.compile(#loss='binary_crossentropy', # 이진분류시 손실함수
              loss="sparse_categorical_crossentropy",
              optimizer='adam',
              metrics=['accuracy'])
begin = time() # 70.1.1부터 현재까지의 초수
hist = model.fit(X_train, y_train,
                 epochs=MY_EPOCH,
                 batch_size=MY_BATCH,
                 validation_split=0.2,
                 verbose=1)
end = time() # 70.1.1부터 이 시점까지의 초수
print('총 학습시간 :', (end-begin))
```