딥러닝 자연어 처리 심화(1)

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

11

APPLICATION

ARTIFICIAL INTELLIGENCE

Notice

THINKING

생각해보기

✓ NLU, NLG, Transformer 모델, Bert 모델은 무엇일까요?

학습목표

GOALS

1 NLU, NLG의 의미를 이해하고 설명할 수 있다.

2 NLU와 NLG의 활용 범위에 이해하고 설명할 수 있다.

3 Transformer 모델의 구조와 구조로 인해 생기는 이점을 이해하고 설명할 수 있다.

4 Bert 모델의 구조와 구조로 인해 생기는 이점을 이해하고 설명할 수 있다.

1 NLU와 NLG

2 고도화된 NLU 모델

3 실습

CONTENTS

학습내용

APPLICATION

01 NLU와 NLG

자연어 이해(NLU)는

자연어 입력을 기계가 이해할 수 있도록 만드는 과정으로서

NLP의 첫번째 컴포넌트

단어, 구, 문장과 같이 언어구조에 대한 이해를 다룸

구문 뿐만 아니라 의미에도 중점

형태학적 분석, 어휘 분석, 구문 분석, 의미 분석, 모호성 처리, 담화 통합, 화용 분석 등이 활용됨

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

자연어 생성(NLG)는

기계를 이용하여 자연어를 생성하는 과정으로서

NLP의 두번째 컴포넌트

사전에 축적되어 있는 조각난 정보들을 모아 사람의 말로 만들어주는 과정이 필요함

사용자의 물음에 대한 답변을 자연스러운 문장으로 만드는 과정에 활용되는 기술임

주로 챗봇에 많이 이용되는 기술임

기존 Seq2Seq가 가지고 있는 bottle neck 문제를 해결하기 위함

2 tb Parmesan cheese — chopped
1 c Coconut milk
3 Eggs, beaten

Place each pasta over layers of lumps. Shape mixture into the moderate oven and simmer until firm. Serve hot in bodied fresh, mustard, orange and cheese.

Combine the cheese and salt together the dough in a large skillet; add the ingredients and stir in the chocolate and pepper.

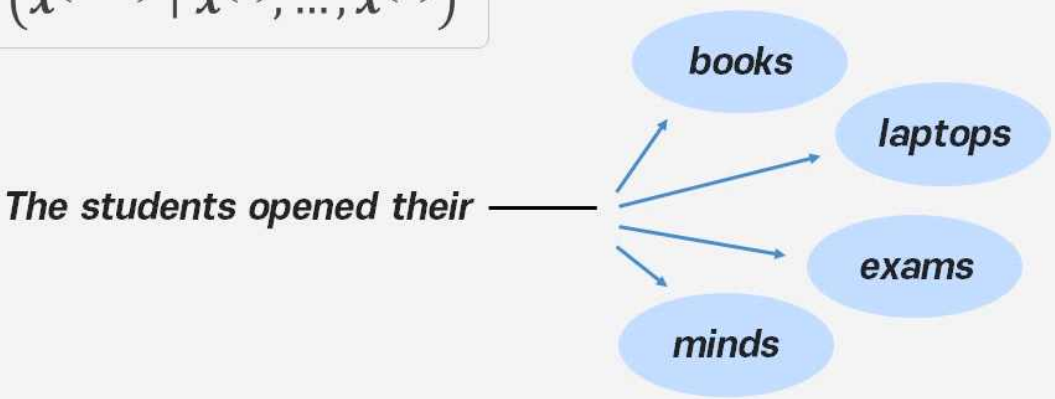
Source: https://gist.github.com/nylki/1efbaa36635956d35bcc

출처: 퍼블릭에이아이(www.publicai.co.kr)
이미지투데이(twi001t1867214_http://www.imagetoday.co.kr/)

아래와 같이 다음 단어를 예측하는 언어 모델을 이용하여 텍스트 생성

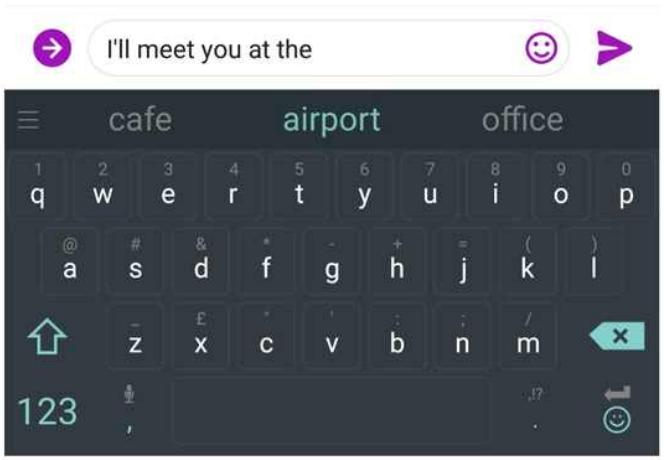
- $x^{(1)}, x^{(2)}, \ldots, x^{(t)}$ 라는 단어 시퀀스가 주어졌을 때 다음 단어인 $x^{(t+1)}$ 의 확률 분포를 계산하여 예측하는 것

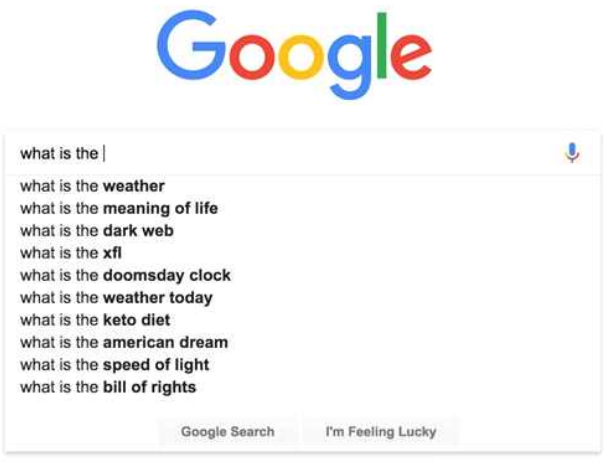
$$P\left(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}\right)$$



04 NLG의 기본 원리 : 언어모델

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING





출처: 퍼블릭에이아이(www.publicai.co.kr)

N개의 연속적인 단어의 말뭉치인 n-gram들의 등장빈도 등의 통계치를 바탕으로 다음 단어를 예측

- 우선 Markov 가정에 의해 다음 단어를 예측

✓ $x^{(t+1)}$은 오직 앞에 나온 n-1개의 단어들에 의해 예측됨

n-1 words

$$P\left(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(1)}\right) = P\left(x^{(t+1)} \mid x^{(t)}, \ldots, x^{(t-n+)2}\right)$$

(assumption)

Prob of n-gram

$$= \frac{P\left(x^{(t+1)}, x^{(t)}, \ldots, x^{(t-n+)2}\right)}{P\left(x^{(t)}, \ldots, x^{(t-n+)2}\right)}$$

Prob of a (n-1)-gram

(definition of Conditional prob)

4-gram Language Model 예시

~~*As the proctor started the clock,*~~ *the students opened their* _____

discard

condition on this

$$P(w \mid \textit{students opened their}) = \frac{\text{count}(\text{students opened their } w)}{\text{count}(\text{students opened their})}$$

- 예를 들어, "students opened their"가 1000번 나오고 "students opened their books"가 400번, students opened their exams"가 100번 나왔으면 books, exams가 나올 각각의 확률은 0.4 , 0.1

N-gram 모델의 한계

- 희소문제

"students opened their (word)"가 data 내에서 한번도 발생하지 않았을 경우엔 (word)가 가지는 확률

0

➡ "students opened their" 문장 자체가 data 내에 한번도 발생하지 않았을 경우에는 word 예측을 위한 계산을 할 수 없게 됨

05 N-gram 모델을 이용한 텍스트 생성

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

N-gram 모델의 한계

N-gram 모델

- 텍스트를 생성할 수 있음
- 대개 문법적으로 맞는 결과를 도출함

- 그러나 일관성이 떨어지는 문제가 발생함

예측할 단어 중 N개의 단어를 들고와서 임베딩 후 모델에 넣어서 다음 단어를 예측하는 방식

output distribution

$$\hat{y} = \text{softmax}(Uh + b_2) \in \mathbb{R}^{|V|}$$

hidden layer

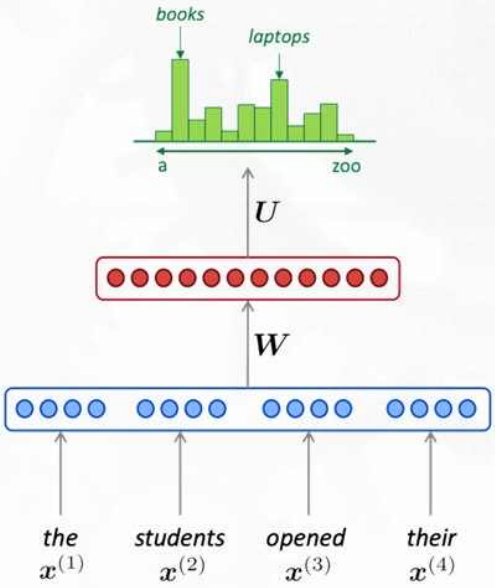
$$h = f(We + b_1)$$

concatenated word embeddings

$$e = [e^{(1)}; e^{(2)}; e^{(3)}; e^{(4)}]$$

words / one-hot vectors

$$x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}$$



출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

06 Fixed window neural network

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

1. N-gram 언어모델이 가졌던 sparsity 문제와 모든 관측된 n-gram을 다 저장해야 해서 생긴 storage 문제도 해결

→ 그러나 fixed window가 여전히 너무 작음

- Window를 크게 하면 W가 너무 커져 계산에 어려움이 생김

2. N-gram 모델과 마찬가지로 다음 단어를 예측할 때 모든 이전 단어를 참고하는 것이 아니라 정해진 n개의 단어만을 참고

기존 모델

- 예측에 참고하는 범위가 정해져 있어 매번 다른 길이의 입력 시퀀스에 대해서 처리하는 능력이 없었음

VS

RNNLM

- Time step을 고려하여 언어모델을 만들면 입력의 길이를 고정하지 않아도 다양한 길이의 입력 시퀀스에 대해 처리가 가능함

07 RNNLM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM의 학습 과정

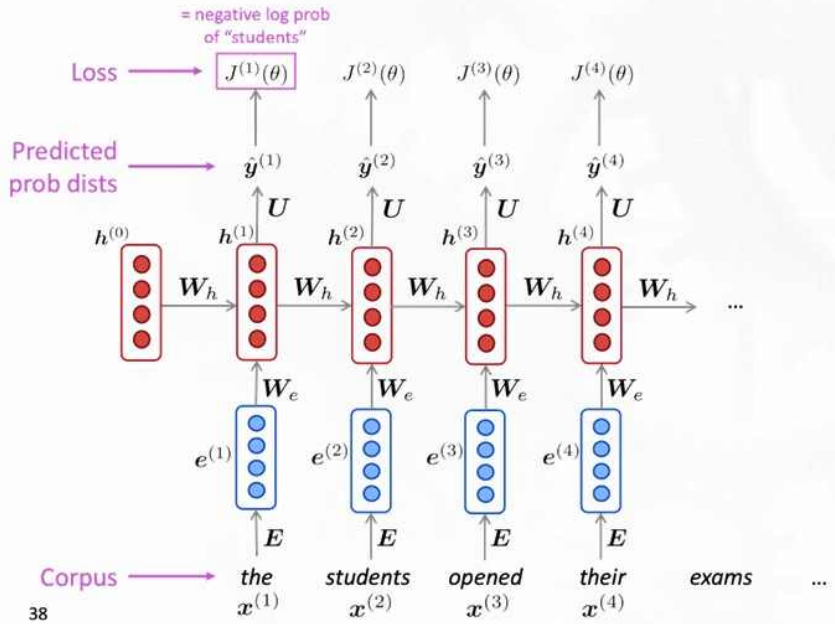
- $\boldsymbol{x}^{(1)}, \boldsymbol{x}^{(2)}, \ldots, \boldsymbol{x}^{(T)}$ 로 이루어진 단어들의 sequence를 RNN-LM모델에 입력하여 매 time step t 마다 $\hat{\boldsymbol{y}}^{(t)}$ 을 계산
- 매 스텝 t마다의 loss function은 예측된 probability distribution $\hat{\boldsymbol{y}}^{(t)}$ 와 실제 다음 단어인 $\boldsymbol{y}^{(t)}$ 와의 cross entropy(negative log probability)

$$J^{(t)}(\theta) = CE\left(\boldsymbol{y}^{(t)}, \hat{\boldsymbol{y}}^{(t)}\right) = -\sum_{w \in V} \boldsymbol{y}_w^{(t)} \log \hat{\boldsymbol{y}}_w^{(t)} = -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

- 총 loss는 이를 평균 낸 값이며, 이 값을 최소화하는 방향으로 학습

$$J(\theta) = \frac{1}{T}\sum_{t=1}^{T} J^{(t)}(\theta) = \frac{1}{T}\sum_{t=1}^{T} -\log \hat{\boldsymbol{y}}_{\boldsymbol{x}_{t+1}}^{(t)}$$

RNNLM의 학습 과정

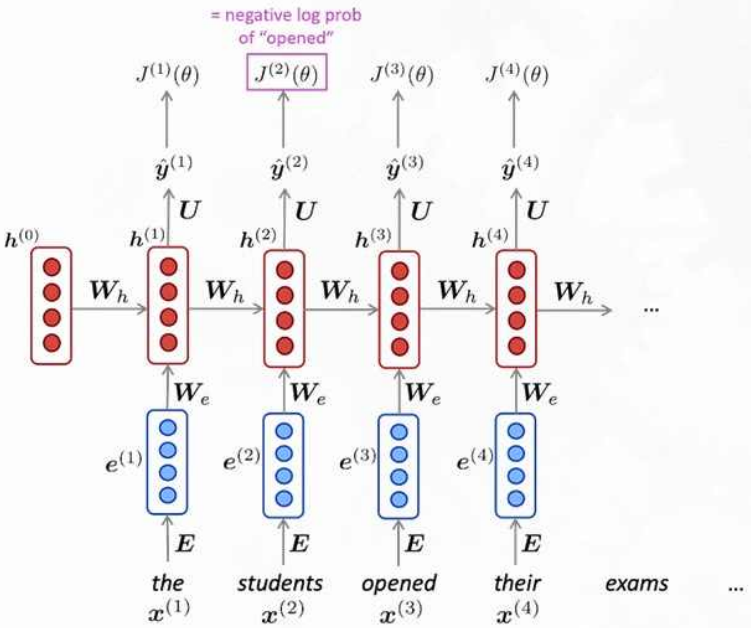


출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

07 RNNLM

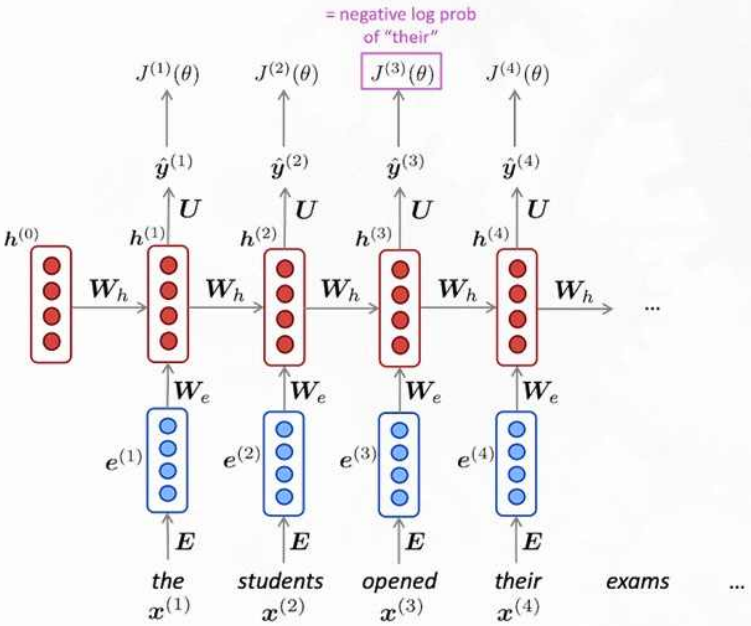
DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM의 학습 과정



출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

RNNLM의 학습 과정

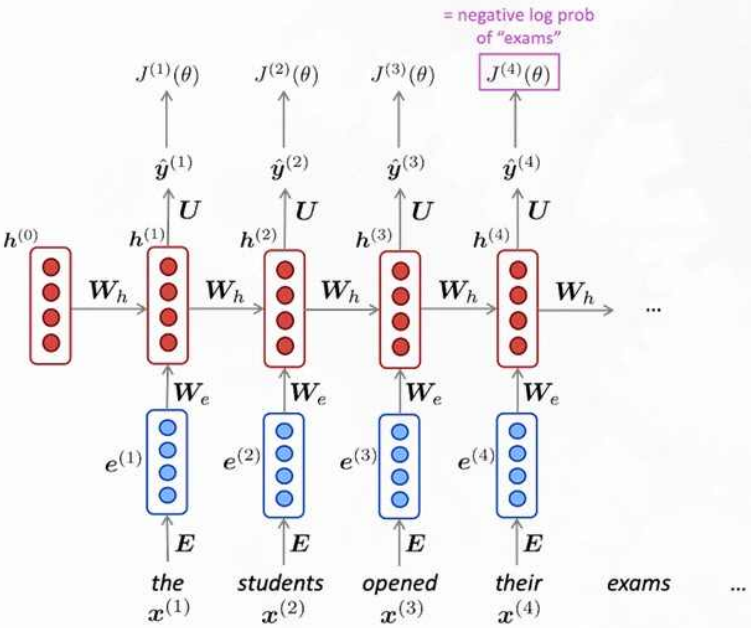


출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

07 RNNLM

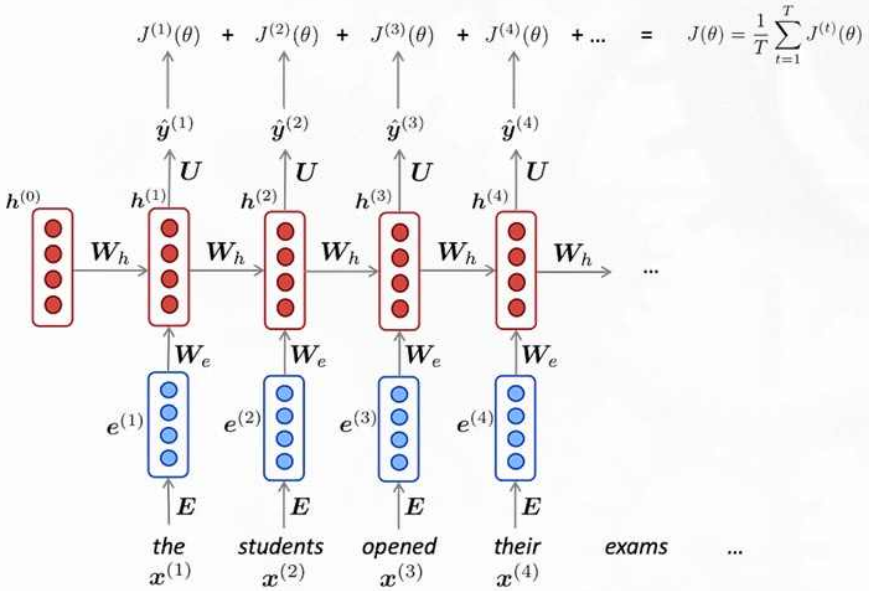
DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM의 학습 과정



출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

RNNLM의 학습 과정



출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

07 RNNLM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM을 이용한 텍스트 생성

- 세가지 문장

'빨간 자동차가 다리를 건너고 있다'

'웰시코기의 다리는 짧다'

'아들은 나에게 새로운 세상으로 건너가는 다리를 놓아주었다'

➡ 모델이 문맥을 학습할 수 있도록 전체 문장의 앞의 단어들을 전부 고려하여 학습하도록 데이터를 재구성 > 12개의 샘플 구성

RNNLM을 이용한 텍스트 생성

Sample	X	y
1	빨간	자동차가
2	빨간 자동차가	다리를
3	빨간 자동차가 다리를	건너고
4	빨간 자동차가 다리를 건너고	있다
5	웰시코기의	다리는
6	웰시코기의 다리는	짧다
7	아들은	나에게
8	아들은 나에게	새로운
9	아들은 나에게 새로운	세상으로
10	아들은 나에게 새로운 세상으로	건너가는
11	아들은 나에게 새로운 세상으로 건너가는	다리를
12	아들은 나에게 새로운 세상으로 건너가는 다리를	놓아주었다

07 RNNLM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM을 이용한 텍스트 생성

- 필요한 라이브러리 import 및 예시문장 저장

```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
from tensorflow.keras.utils import to_categorical
text = """빨간 자동차가 다리를 건너고 있다\n
웰시코기의 다리는 짧다\n
아들은 나에게 새로운 세상으로 건너가는 다리를 놓아주었다\n"""
```

RNNLM을 이용한 텍스트 생성

- 단어 집합 생성 후 크기 확인

```
t = Tokenizer()
t.fit_on_texts([text])
vocab_size = len(t.word_index)+1
# 케라스 토크나이저의 정수 인코딩은 인덱스가 1부터 시작하지만,
# 케라스 원-핫 인코딩에서 배열의 인덱스가 0부터 시작하기 때문에
# 배열의 크기를 실제 단어 집합의 크기보다 +1로 생성해야하므로 미리 +1 선언
print('단어 집합의 크기 : %d' % vocab_size)
```

07 RNNLM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM을 이용한 텍스트 생성

- 전체 샘플 출력해서 확인 후 패딩을 통해 가장 긴 샘플의 길이를 기준으로 샘플 길이 통일

```
print(sequences)
>>[[2, 3], [2, 3, 1], [2, 3, 1, 4], [2, 3, 1, 4, 5], [6, 1], [6, 1, 7], [8,
1], [8, 1, 9], [8, 1, 9, 10], [8, 1, 9, 10, 1], [8, 1, 9, 10, 1, 11]]
max_len=max(len(l) for l in sequences) # 모든 샘플에서 길이가 가장 긴 샘플의 길이 출력
print('샘플의 최대 길이 : {}'.format(max_len))
>>샘플의 최대 길이 : 7
sequences = pad_sequences(sequences, maxlen=max_len, padding='pre')
```

RNNLM을 이용한 텍스트 생성

- 패딩된 훈련 데이터 확인

```
print(sequences)
>>[[ 0  0  0  0  0  2  3]
 [ 0  0  0  0  2  3  1]
 [ 0  0  0  2  3  1  4]
 [ 0  0  2  3  1  4  5]
 [ 0  0  0  0  0  6  7]
 [ 0  0  0  0  6  7  8]
 [ 0  0  0  0  0  9 10]
 [ 0  0  0  0  9 10 11]
 [ 0  0  0  9 10 11 12]
 [ 0  0  9 10 11 12 13]
 [ 0  9 10 11 12 13  1]
 [ 9 10 11 12 13  1 14]]
```

07 RNNLM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM을 이용한 텍스트 생성

- 각 샘플의 마지막 단어를 레이블로 분리

```
sequences = np.array(sequences)
X = sequences[:,:-1]
y = sequences[:,-1]
# 리스트의 마지막 값을 제외하고 저장한 것은 X
# 리스트의 마지막 값만 저장한 것은 y. 이는 레이블에 해당됨.
```

RNNLM을 이용한 텍스트 생성

- 분리된 X, y확인, y는 one-hot encoding

```
print(X)
>>[[ 0  0  0  0  0  2]
 [ 0  0  0  0  2  3]
 [ 0  0  0  2  3  1]
 [ 0  0  2  3  1  4]
 [ 0  0  0  0  0  6]
 [ 0  0  0  0  6  7]
 [ 0  0  0  0  0  9]
 [ 0  0  0  0  9 10]
 [ 0  0  0  9 10 11]
 [ 0  0  9 10 11 12]
 [ 0  9 10 11 12 13]
 [ 9 10 11 12 13  1]]
```

```
print(y)
>>[ 3  1  4  5  7  8 10 11 12 13  1 14]
y = to_categorical(y, num_classes=vocab_size)
print(y)
>>[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
 [0. 1. 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. 1.]]
```

07 RNNLM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM을 이용한 텍스트 생성

- RNN 모델에 데이터 train

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, SimpleRNN

model = Sequential()
model.add(Embedding(vocab_size, 10, input_length=max_len-1)) # 레이블을 분리하였으므로 이제 X의 길이는 5
model.add(SimpleRNN(32))
model.add(Dense(vocab_size, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
model.fit(X, y, epochs=200, verbose=2)
```

RNNLM을 이용한 텍스트 생성

- 문장 생성 함수

```
def sentence_generation(model, t, current_word, n): # 모델, 토크나이저, 현재 단어, 반복할 횟수
    init_word = current_word # 처음 들어온 단어도 마지막에 같이 출력하기위해 저장
    sentence = ''
    for _ in range(n): # n번 반복
        encoded = t.texts_to_sequences([current_word])[0] # 현재 단어에 대한 정수 인코딩
        encoded = pad_sequences([encoded], maxlen=5, padding='pre') # 데이터에 대한 패딩
        result = model.predict_classes(encoded, verbose=0)
    # 입력한 X(현재 단어)에 대해서 Y를 예측하고 Y(예측한 단어)를 result에 저장.
```

07 RNNLM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

RNNLM을 이용한 텍스트 생성

- 문장 생성 함수

```
        for word, index in t.word_index.items():
            if index == result: # 만약 예측한 단어와 인덱스와 동일한 단어가 있다면
                break # 해당 단어가 예측 단어이므로 break
        current_word = current_word + ' '  + word # 현재 단어 + ' ' + 예측 단어를 현재 단어로 변경
        sentence = sentence + ' ' + word # 예측 단어를 문장에 저장
    # for문이므로 이 행동을 다시 반복
    sentence = init_word + sentence
    return sentence
```

RNNLM을 이용한 텍스트 생성

- 문장 생성함수를 통한 결과 확인

```
print(sentence_generation(model, t, '빨간', 4))
# '빨간' 라는 단어 x뒤에는 총 4개의 단어가 있으므로 4번 예측
>>빨간 자동차가 다리를 건너고 있다
print(sentence_generation(model, t, '웰시코기의', 2)) # 2번 예측
>>웰시코기의 다리는 짧다
print(sentence_generation(model, t, '아들은', 6)) # 6번 예측
>>아들은 나에게 새로운 세상으로 건너가는 다리를 놓아주었다
```


LSTM을 이용한 텍스트 생성

- 필요한 라이브러리 import

```
import pandas as pd
from string import punctuation
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
import numpy as np
from tensorflow.keras.utils import to_categorical
```

LSTM을 이용한 텍스트 생성

- 데이터 불러와 5개 출력해 보기

```
df=pd.read_csv('archive/ArticlesApril2018.csv')
df.head()
# https://www.kaggle.com/aashita/nyt-comments
headline = [] # 리스트 선언
headline.extend(list(df.headline.values)) # 헤드라인의 값들을 리스트로 저장
headline[:5] # 상위 5개만 출력
```

08 LSTM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

LSTM을 이용한 텍스트 생성

- 샘플에 노이즈 확인 후 제거

```
print('총 샘플의 개수 : {}'.format(len(headline))) #현재 샘플의 개수
>>총 샘플의 개수 : 1324
headline = [n for n in headline if n != "Unknown"] # Unknown 값을 가진 샘플 제거
print('노이즈값 제거 후 샘플의 개수 : {}'.format(len(headline))) # 제거 후 샘플의 개수
>>노이즈값 제거 후 샘플의 개수 : 1214
headline[:5]
```

LSTM을 이용한 텍스트 생성

- 구두점 제거 및 소문자화

```
def repreprocessing(s):
    s=s.encode("utf8").decode("ascii",'ignore')
    return ''.join(c for c in s if c not in punctuation).lower() # 구두점 제거와 동시에 소문자화

text = [repreprocessing(x) for x in headline]
text[:5]
```

08 LSTM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

LSTM을 이용한 텍스트 생성

- 단어 집합 생성 및 크기 확인

```
t = Tokenizer()
t.fit_on_texts(text)
vocab_size = len(t.word_index) + 1
print('단어 집합의 크기 : %d' % vocab_size)
```

LSTM을 이용한 텍스트 생성

- 정수 인코딩과 동시에 하나의 문장을 여러 줄로 분해하여 훈련 데이터 구성

```
sequences = list()

for line in text: # 1,214 개의 샘플에 대해서 샘플을 1개씩 가져온다.
    encoded = t.texts_to_sequences([line])[0] # 각 샘플에 대한 정수 인코딩
    for i in range(1, len(encoded)):
        sequence = encoded[:i+1]
        sequences.append(sequence)

sequences[:11] # 11개의 샘플 출력
```

08 LSTM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

LSTM을 이용한 텍스트 생성

- 인덱스로부터 단어를 찾는 딕셔너리 생성

```
index_to_word={}
for key, value in t.word_index.items(): # 인덱스를 단어로 바꾸기 위해 index_to_word를 생
성
    index_to_word[value] = key

print('빈도수 상위 582번 단어 : {}'.format(index_to_word[582]))
```

LSTM을 이용한 텍스트 생성

- 가장 긴 길 샘플의 길이 기준으로 모든 샘플의 길이 패딩

```
max_len=max(len(l) for l in sequences)
print('샘플의 최대 길이 : {}'.format(max_len))
sequences = pad_sequences(sequences, maxlen=max_len, padding='pre')
print(sequences[:3])
```

08 LSTM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

LSTM을 이용한 텍스트 생성

- X, y 분리 및 y one-hot encoding

```
sequences = np.array(sequences)
X = sequences[:,:-1]
y = sequences[:,-1]
y = to_categorical(y, num_classes=vocab_size)
```

LSTM을 이용한 텍스트 생성

- 모델 학습

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Dense, LSTM

model = Sequential()
model.add(Embedding(vocab_size, 10, input_length=max_len-1))
# y데이터를 분리하였으므로 이제 X데이터의 길이는 기존 데이터의 길이 - 1
model.add(LSTM(128))
model.add(Dense(vocab_size, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=
['accuracy'])
model.fit(X, y, epochs=200, verbose=2)
```

08 LSTM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

LSTM을 이용한 텍스트 생성

- 문장 생성 함수

```
def sentence_generation(model, t, current_word, n): # 모델, 토크나이저, 현재 단어, 반복
할 횟수
    init_word = current_word # 처음 들어온 단어도 마지막에 같이 출력하기위해 저장
    sentence = ''
    for _ in range(n): # n번 반복
        encoded = t.texts_to_sequences([current_word])[0] # 현재 단어에 대한 정수 인코
딩
        encoded = pad_sequences([encoded], maxlen=23, padding='pre') # 데이터에 대
한 패딩
        result = model.predict_classes(encoded, verbose=0)
    # 입력한 X(현재 단어)에 대해서 y를 예측하고 y(예측한 단어)를 result에 저장.
```

LSTM을 이용한 텍스트 생성

- 문장 생성 함수

```
        for word, index in t.word_index.items():
            if index == result: # 만약 예측한 단어와 인덱스와 동일한 단어가 있다면
                break # 해당 단어가 예측 단어이므로 break
        current_word = current_word + ' '  + word # 현재 단어 + ' ' + 예측 단어를 현재
단어로 변경
        sentence = sentence + ' ' + word # 예측 단어를 문장에 저장
    # for문이므로 이 행동을 다시 반복
    sentence = init_word + sentence
    return sentence
```

08 LSTM

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

LSTM을 이용한 텍스트 생성

- 문장 생성 함수

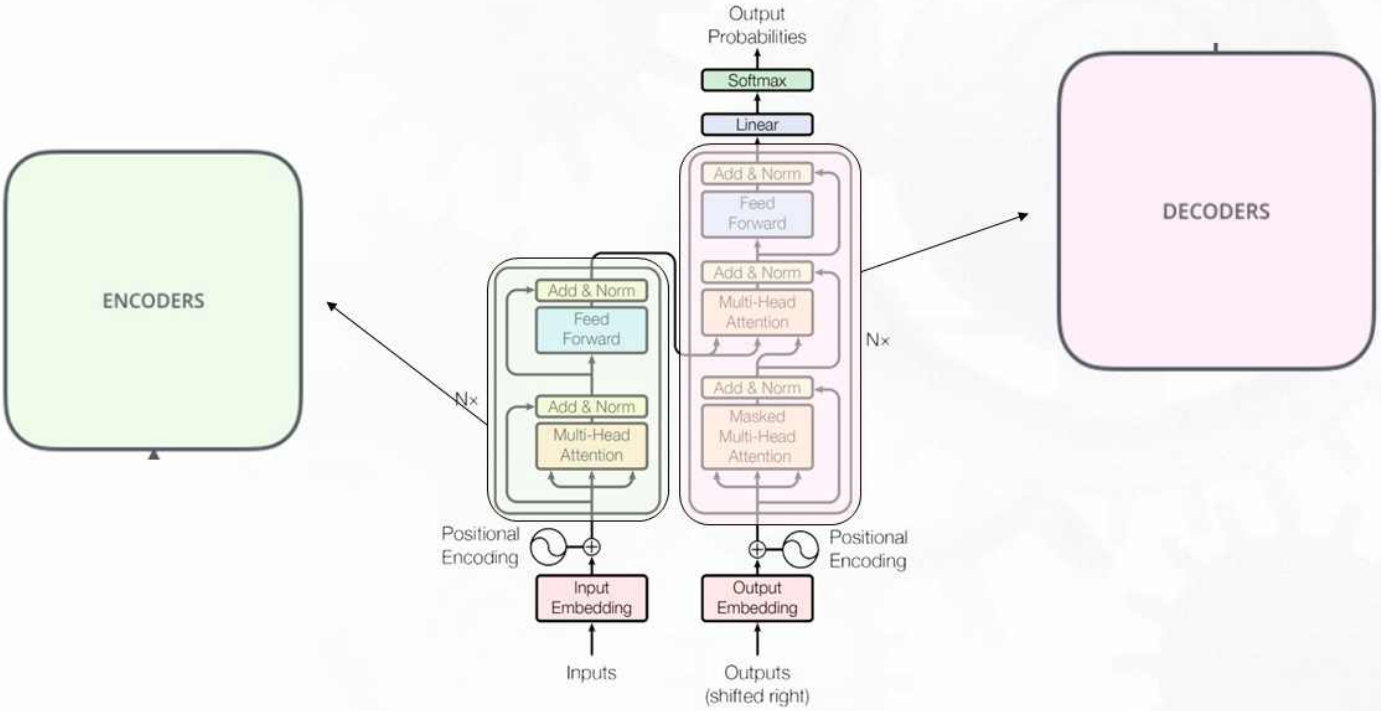
```
print(sentence_generation(model, t, 'i', 10))
# 임의의 단어 'i'에 대해서 10개의 단어를 추가 생성
>>i disapprove of school vouchers can i still apply for them
print(sentence_generation(model, t, 'how', 10))
# 임의의 단어 'how'에 대해서 10개의 단어를 추가 생성
>>how to serve a deranged tyrant stoically home a cold war
```

APPLICATION

02 고도화된 NLU 모델

01 RNN 기반 모델의 한계

1. 모든 Time step을 고려하여 연산하는 RNN 모델의 특성상, 모델이 크며, 연산량이 많고 비용이 많이 듦
2. 순차적인 연산방식으로 인해 멀리 떨어진 정보를 잘 학습하지 못함 (long-term dependency)
3. RNN을 이용하지 않고 self-attention과 positional encoding을 통해 위치정보를 보존하여 학습한다면, 순차적인 방식이 아니라 병렬적인 방식으로 연산하여 연산 속도를 크게 향상 시킬 수 있음



출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 주요 파라미터

d_{mode}

- 트랜스포머의 인코더와 디코더에서의 정해진 입력과 출력의 크기

num_layers

- 트랜스포머 모델에서 인코더와 디코더가 구성된 층의 개수

Transformer 주요 파라미터

num_heads

- 트랜스포머에서 병렬로 수행한 어텐션의 개수

d_{ff}

- 트랜스포머 내부 feed forward neural network의 은닉층 개수

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

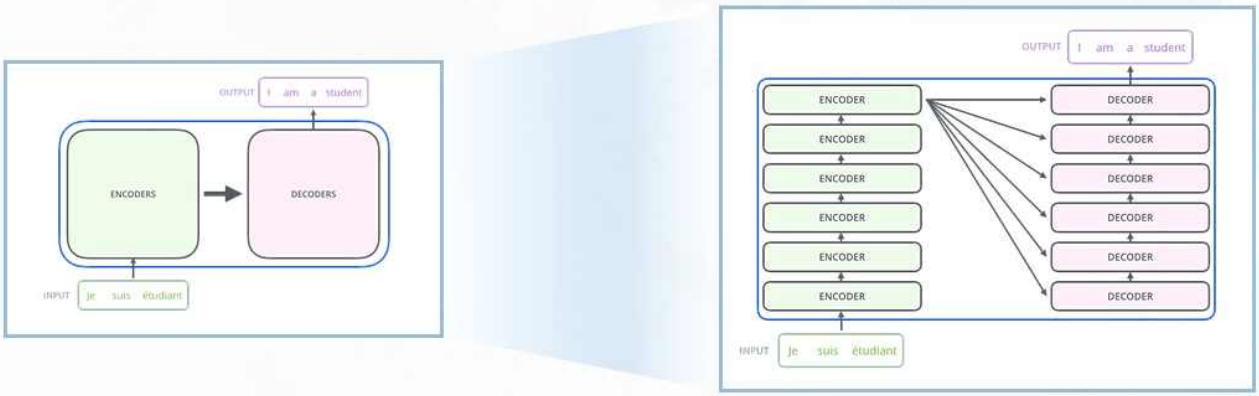
Transformer 기본 구조

- 트랜스포머는 RNN을 사용하지 않지만 기존의 seq2seq처럼 인코더에서 input sequence를 받고 디코더에서 output을 내는 인코더-디코더 구조를 유지

출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조

- 기존 seq2seq : encoder와 decoder에서 각각 하나의 RNN이 t개의 time-step을 가지는 구조
- Transformer : 인코더와 디코더라는 단위가 N개로 구성되는 구조



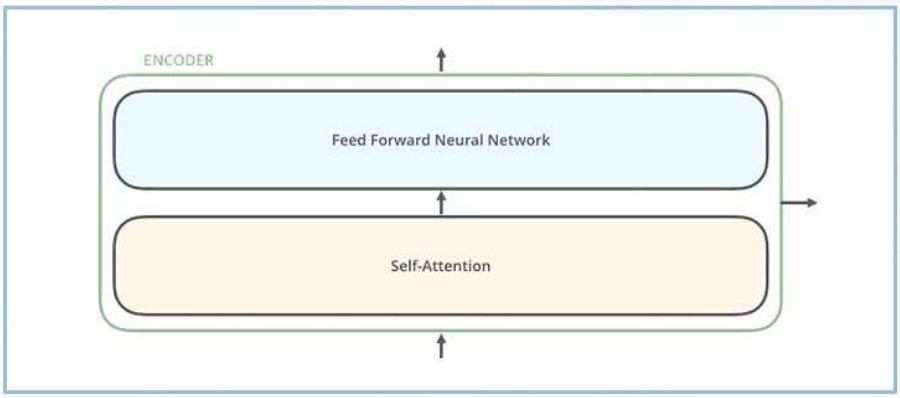
출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

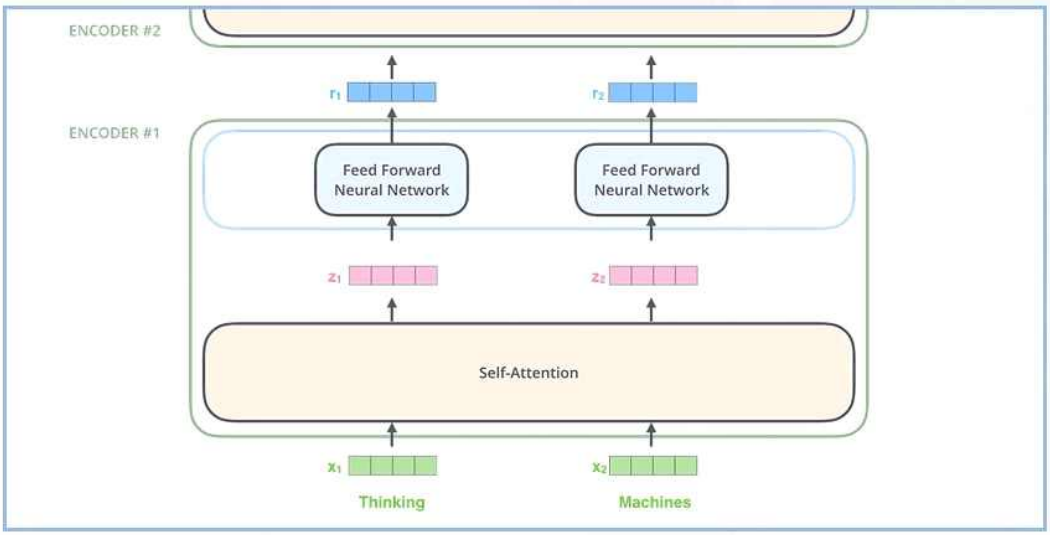
- 트랜스포머는 n개의 인코더 층을 쌓는 데 이 때 n은 하이퍼파라미터로서 인코더층과 디코더층의 개수를 의미
- 인코더층은 크게 총 2개의 sublayer로 구성
  : self-attention(scaled dot product attention), feed-forward neural network



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - encoder

- 각각의 입력 단어들은 embedding 알고리즘을 통해 임베딩된 후 두개의 Sublayer를 통과하여 다음 encoder에 전달됨



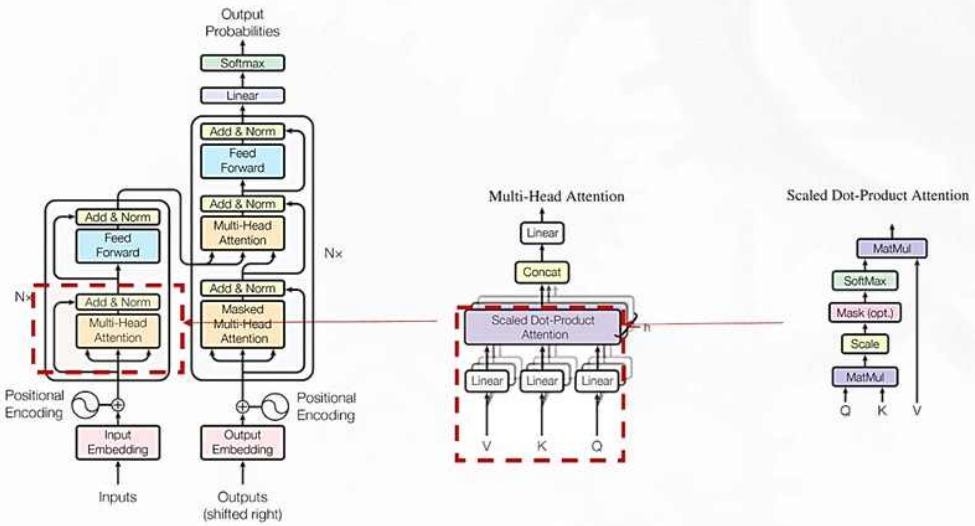
출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

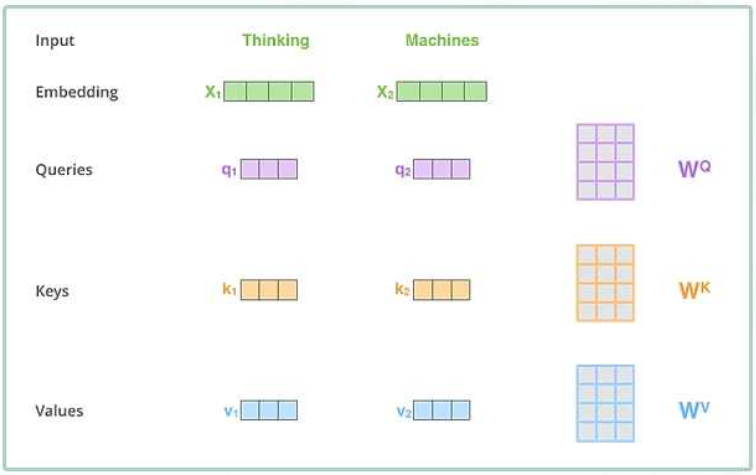


출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

1) 각 input으로부터 3개의 벡터인 q, k, v를 만듦



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

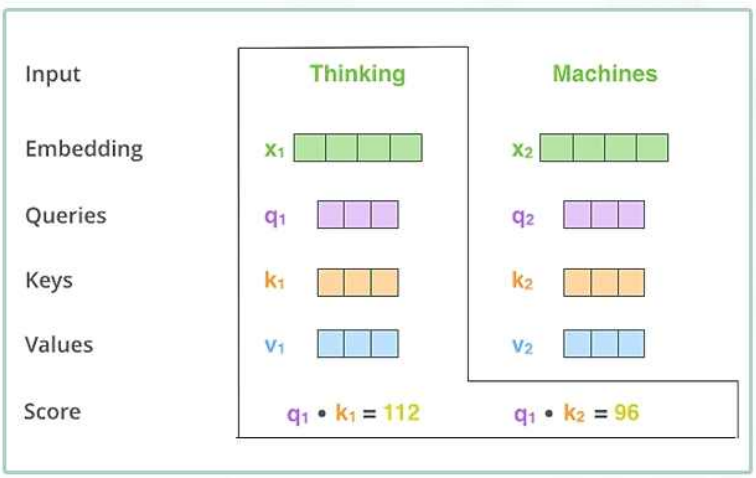
02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

2) 특정 단어와 문장에 사용된 모든 단어와의 점수(연관성) 계산

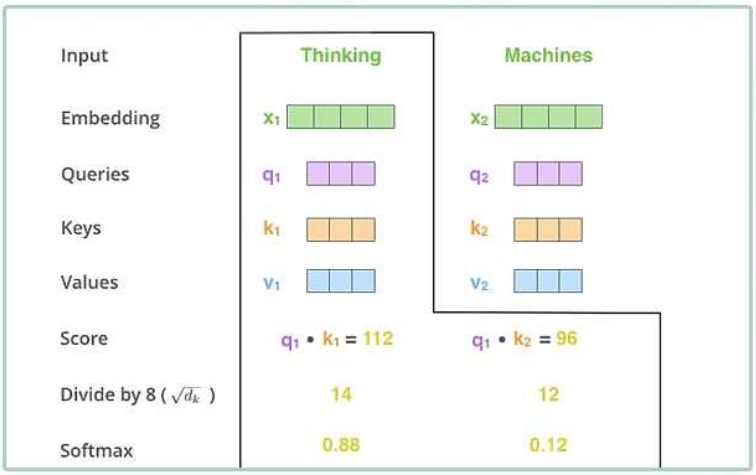


출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

3) 점수를 key vector 길이의 제곱근으로 나누고 softmax



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

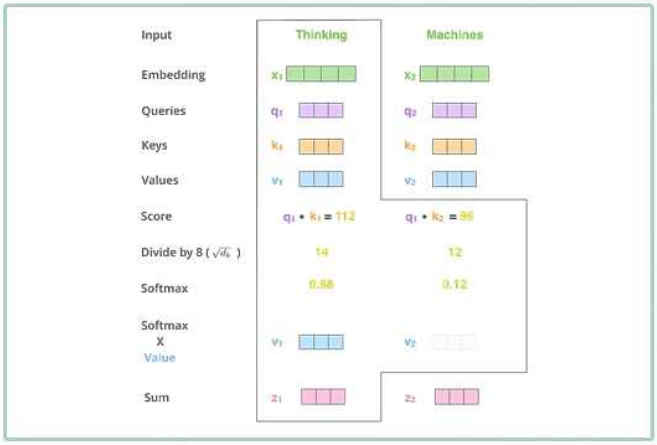
02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

4) 결과값을 각 input의 value vector와 곱하고 이렇게 구한 모든 weighted value vector 들을 더함

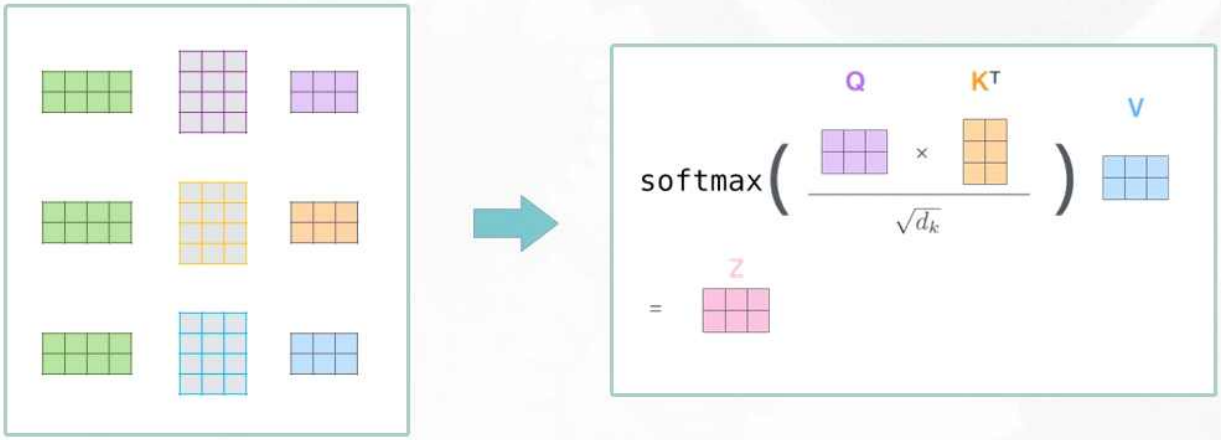


출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ 실제 계산은 행렬 곱으로 이루어짐



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

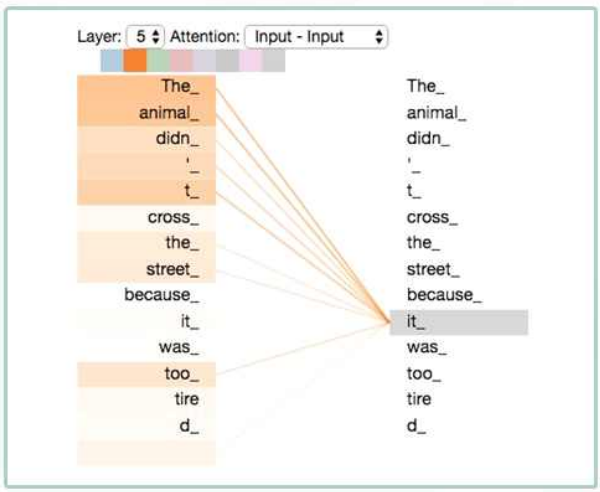
02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ Multi-headed attention

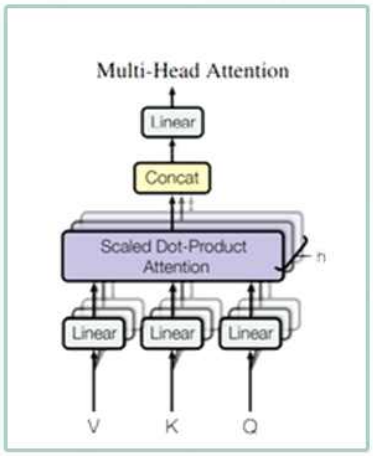


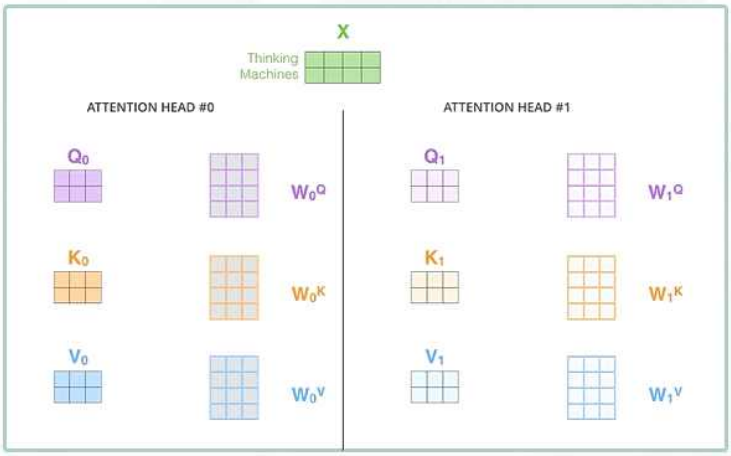
출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ Multi-headed attention





출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/
Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

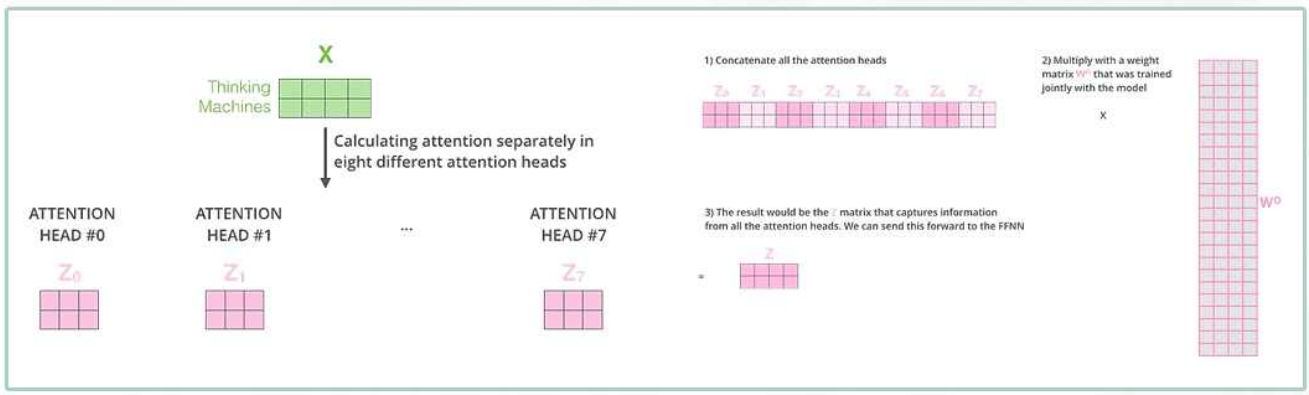
02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ Multi-headed attention

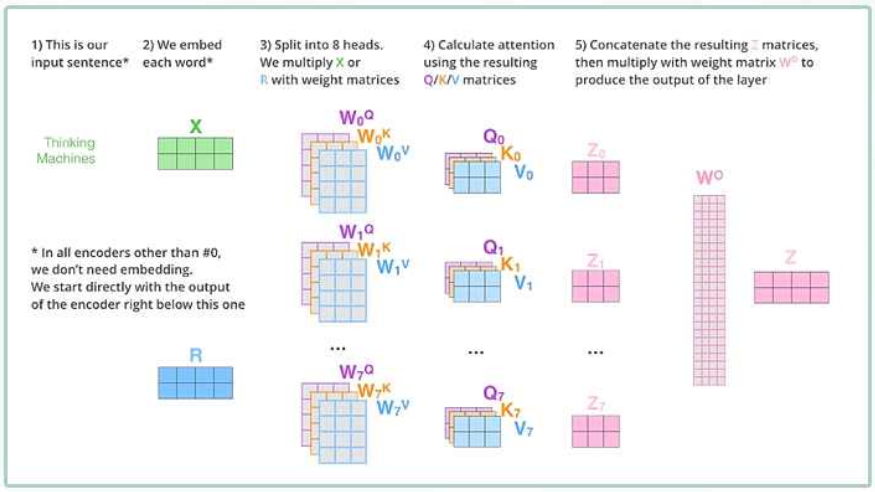


출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ Multi-headed attention



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

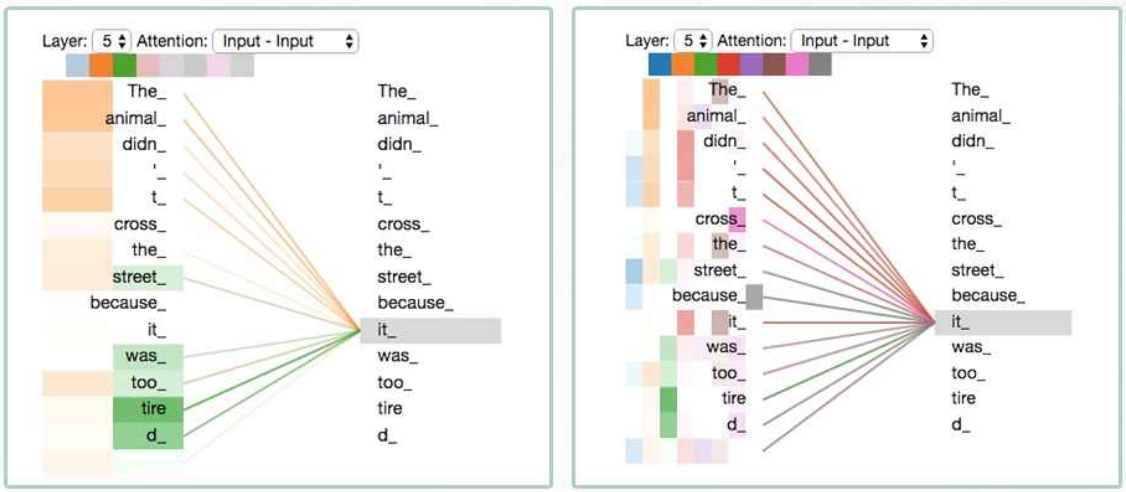
02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ Multi-headed attention

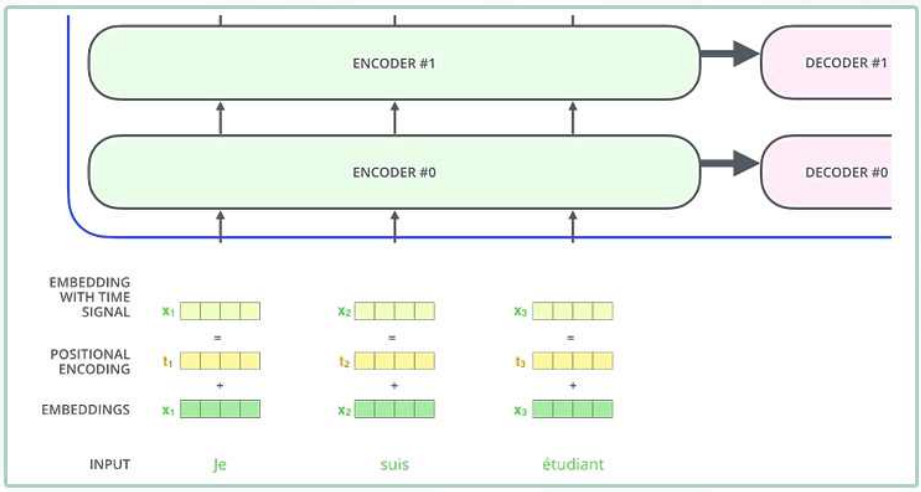


출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ Positional Encoding



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

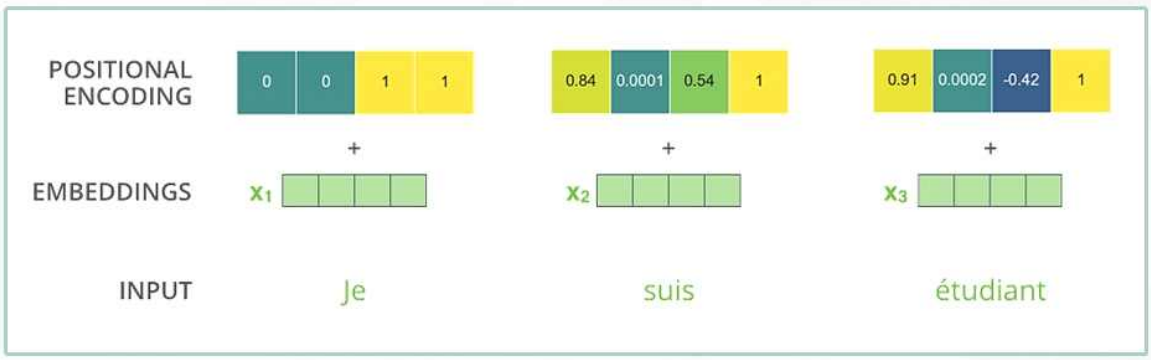
02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ Positional Encoding

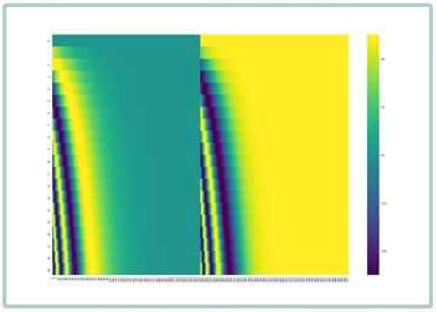


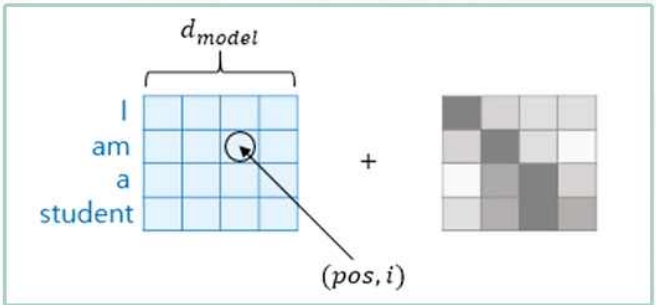
출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - encoder

1 self-attention(scaled dot product attention)

✓ Positional Encoding





$$PE_{(pos,\,2i)} = \sin(\text{pos}/100000^{2i/d}\text{model})$$

$$PE_{(pos,\,2i+1)} = \cos(\text{pos}/100000^{2i/d}\text{model})$$

출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

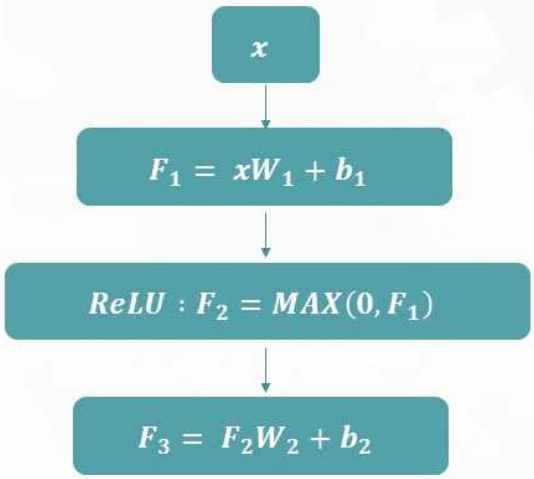
02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

2 Position-wise feed forward net

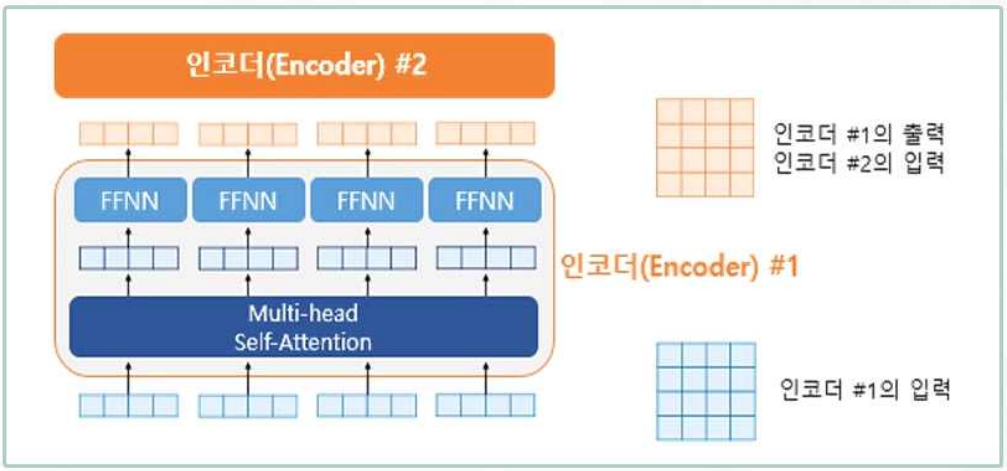
✓ $FFNN(x)=MAX(0,\ xW_1+b_1)\ W_2+b_2$



Transformer 기본 구조 - encoder

2 Position-wise feed forward net

✓ $FFNN(x)=MAX(0,\ xW_1+b_1)\ W_2+b_2$



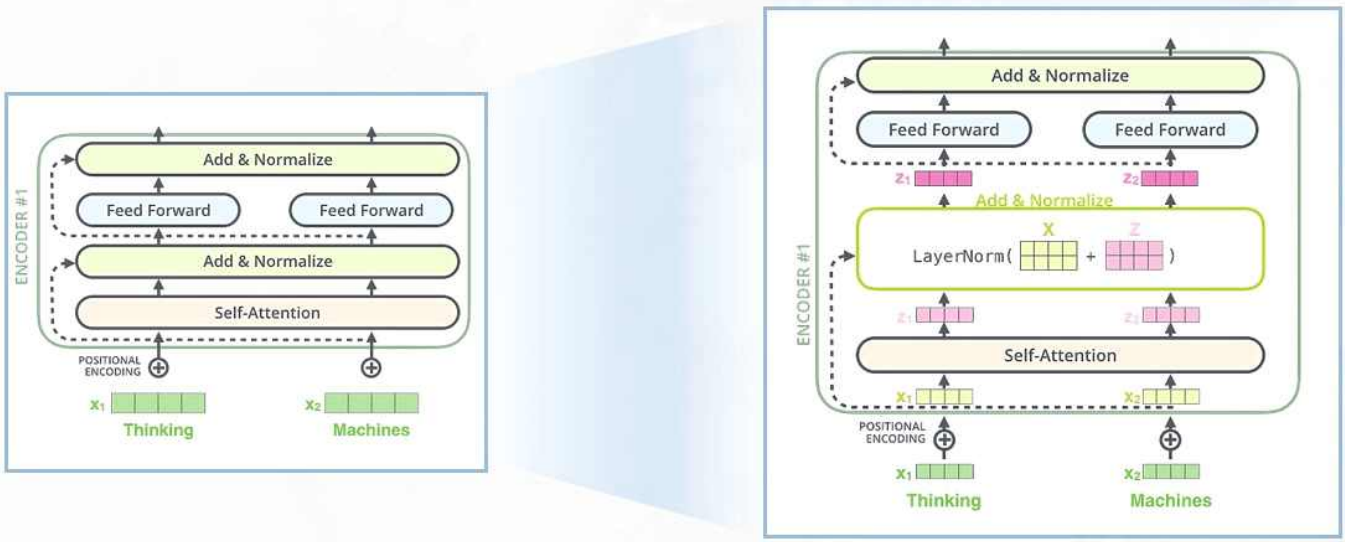
출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

3 Residual connection & Layer normalization

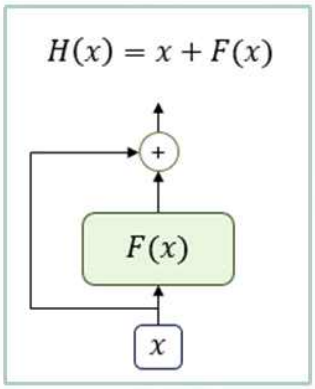


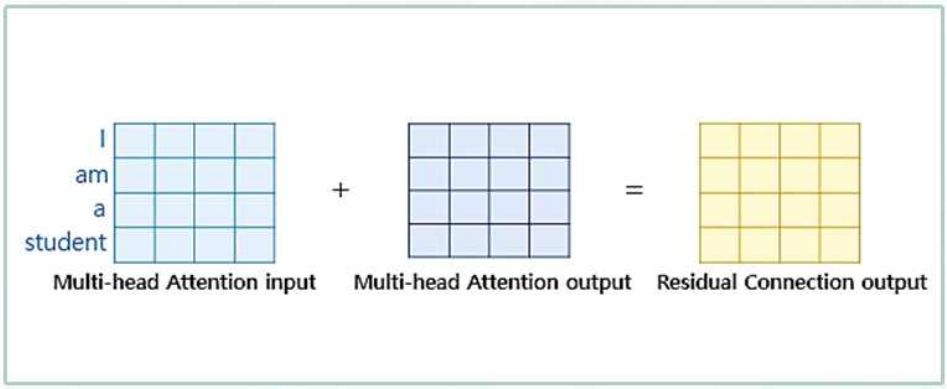
출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - encoder

3 Residual connection & Layer normalization

✓ Residual Connection





$x + Sublayer(x)$

$H(x) = x + Multi\ head\ attention(x)$

출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - encoder

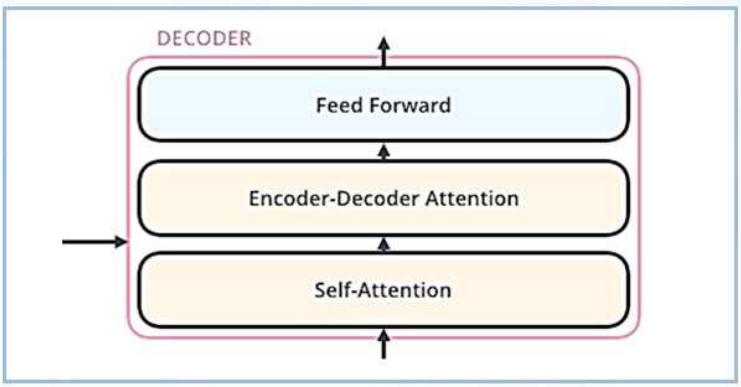
3 Residual connection & Layer normalization

✓ Layer normalization : $LayerNorm(x+Sublayer(x))$

Transformer 기본 구조 - decoder

- 디코더층은 크게 총 3개의 sublayer로 구성

self-attention | encoder-decoder attention | feed-forward neural network



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 기본 구조 - decoder

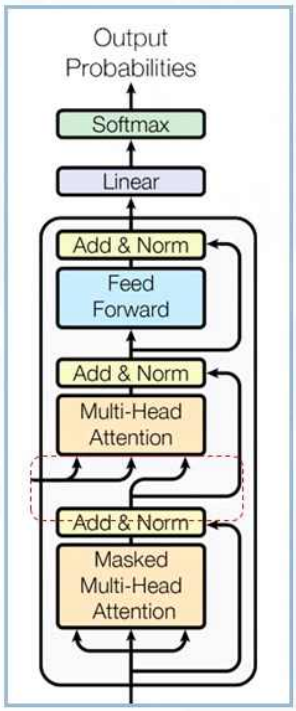
1 self-attention(masked multi-head attention)

- ✓ Decoder는 현재까지 알려진 정보를 바탕으로 새로운 문장을 생성하는 역할을 함
- ✓ 따라서 현재 시점 이후의 시점에 대한 정보를 사용할 수 없어야 하기에, attention 시 제약을 거는 과정을 masking이라 함
- ✓ 보통 현재 시점 이후의 위치에 대해서 -inf 처리로 masking함

Transformer 기본 구조 - decoder

2 encoder-decoder attention(multi-head attention)

- ✓ Seq2Seq에서 encoder hidden states h_j를 이용한 것처럼, Transformer에서도 Encoder의 마지막 layer의 output sequence값을 key, value로 이용
- ✓ Encoder-decoder attention은 decoder가 x_i 의 정보를 표현하기 위해 input sequence의 item j의 정보를 얼마나 이용할지 결정하는 역할



출처: Attention mechanism in NLP From seq2seq + attention to BERT : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

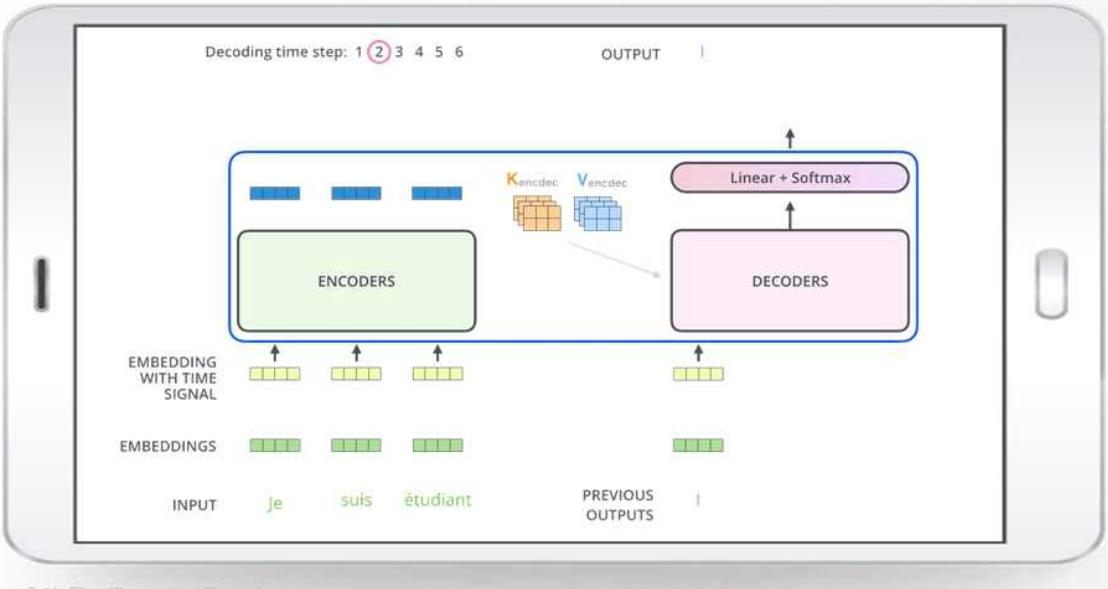
Transformer 기본 구조 - decoder

3 feed forward neural network

- ✓ Decoder self-attention의 결과에 encoder-decoder attention의 결과가 더해져 feed-forward neural network에 입력

Transformer 기본 구조 - decoder

- 전체 작동 구조 – encoding step



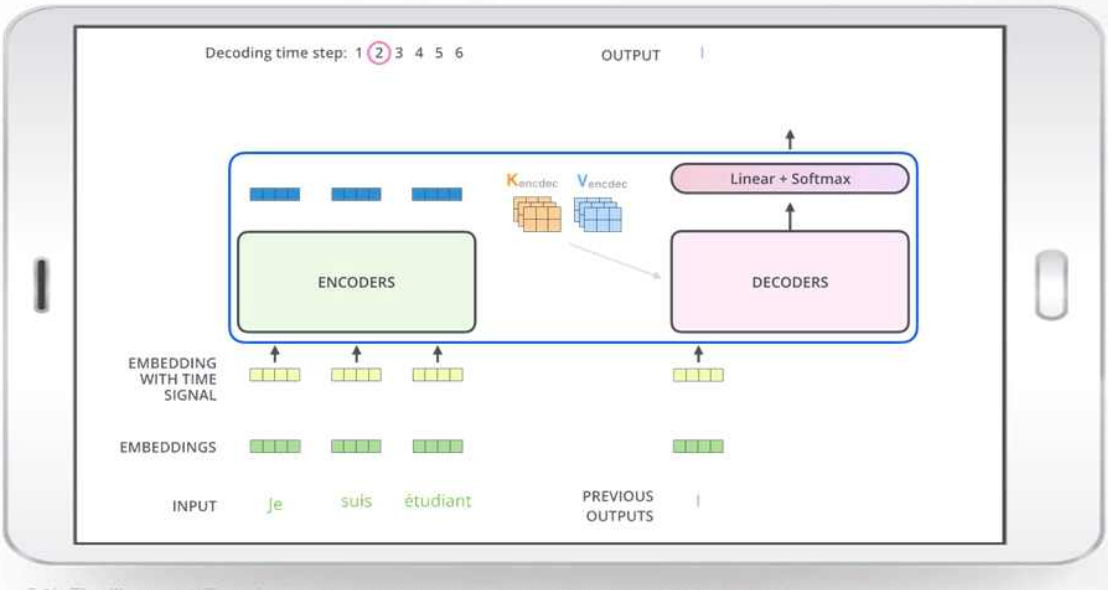
출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

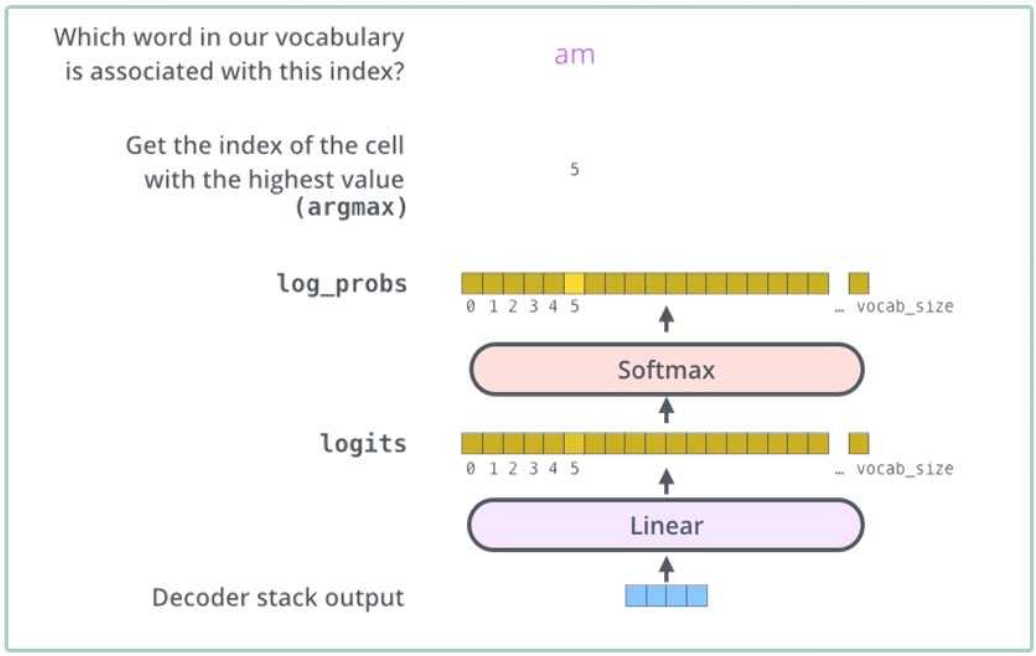
Transformer 기본 구조 - decoder

- 전체 작동 구조 – decoding step



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 기본 구조 - linear&softmax



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 학습 과정

- 라벨링된 학습 데이터 셋과 실제 label과의 비교를 통해 학습
- 예를 들어 데이터 내 ((“a”, “am”, “i”, “thanks”, “student”, and “⟨eos⟩” (‘end of sentence’의 줄임말))) 6개 단어가 있다고 가정

Output Vocabulary

WORD	a	am	I	thanks	student	<eos>
INDEX	0	1	2	3	4	5

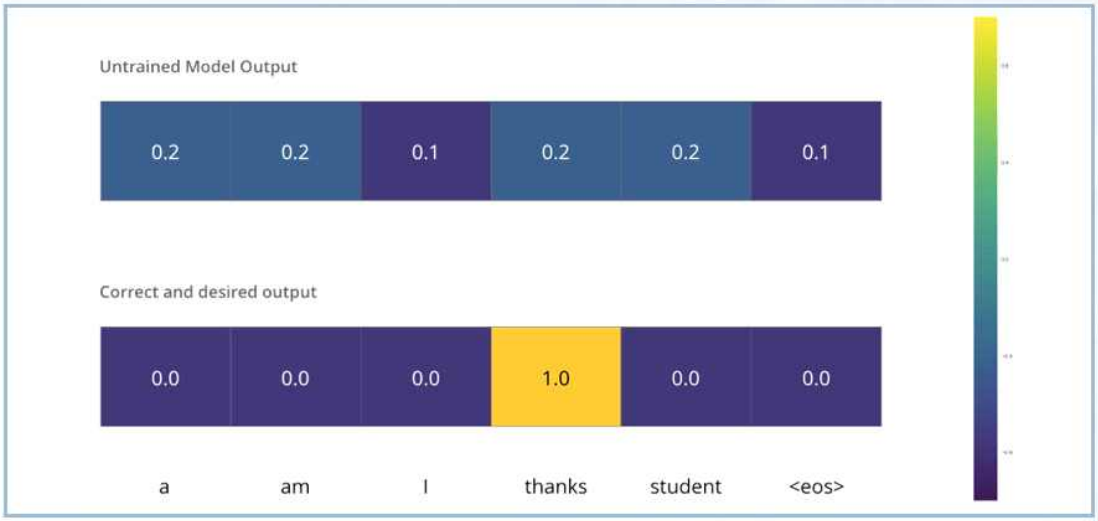
One-hot encoding of the word “am”

0.0	1.0	0.0	0.0	0.0	0.0

출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer 학습 과정

- Loss function : cross entropy
- 예를 들어 “merci”를 “thanks”로 번역하는 상황을 가정



출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

02 Transformer

DEEP LEARNING AND NATURAL LANGUAGE PROCESSING

Transformer 학습 과정

- 예를 들어 불어 문장 “je suis étudiant”을 영어 문장 “i am a student”으로 번역하는 상황 가정

출처: The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/

Transformer의 장점

- Transformer 는 다른 모델들보다 패러매터의 숫자가 적고, feed-forward를 이용하기 때문에 병렬화가 쉬움
- 빠른 연산이 가능함
- 멀리 떨어진 단어 간의 정보가 곧바로 연결되기 때문에 정확한 모델링도 가능함

APPLICATION

03 실습

SUMMARY

학습정리

- NLU, NLG 차이
- Text Generation 기본 원리 이해
- Text Generation 구현 방식 이해
- Transformer 학습 과정
- Transformer 장점 이해

EXPANSION

확장하기

1. NLU와 NLG 각각 무엇을 의미할까요?
2. RNNLM의 기존 모델들과의 차이점에는 어떤 것이 있을까요?
3. 텍스트를 생성하는 방법에는 어떤 것들이 있을까요?
4. RNN 기반 모델의 한계는 무엇일까요?
5. Transformer모델의 주요 특징은 무엇일까요?

참고 문헌

REFERENCE

◆ 참고 사이트
- 용어들에 대한 정의 : https://ko.wikipedia.org/wiki.
- The Illustrated Transformer : http://jalammar.github.io/illustrated-transformer/
- 딥러닝을 이용한 자연어 처리 입문 : https://wikidocs.net/book/2155
- Attention mechanism in NLP From seq2seq + attention to BERT
  : https://lovit.github.io/machine%20learning/2019/03/17/attention_in_nlp/
- 어텐션 메커니즘과 transformer
  : https://medium.com/platfarm/%EC%96%B4%ED%85%90%EC%85%98-%EB%A9%94%EC%BB%A4%EB%8B%88%EC%A6%98%EA%B3%BC-transfomer-self-attention-842498fd3225

◆ 참고 서적
- 잘라지 트하나키, 『파이썬 자연어 처리의 이론과 실제』, 에이콘, 2017

서체 출처 : 에스코어드림체-㈜에스코어, 나눔글꼴체-㈜네이버