

RNN이란? & NLP 기초개념 (Recurrent Neural Network)

KUBIG 12기 이나윤

INDEX

2021년 여름방학 딥러닝 분반 5주차

01

RNN의 개념
& LSTM
& GRU

02

NLP 4가지 단계

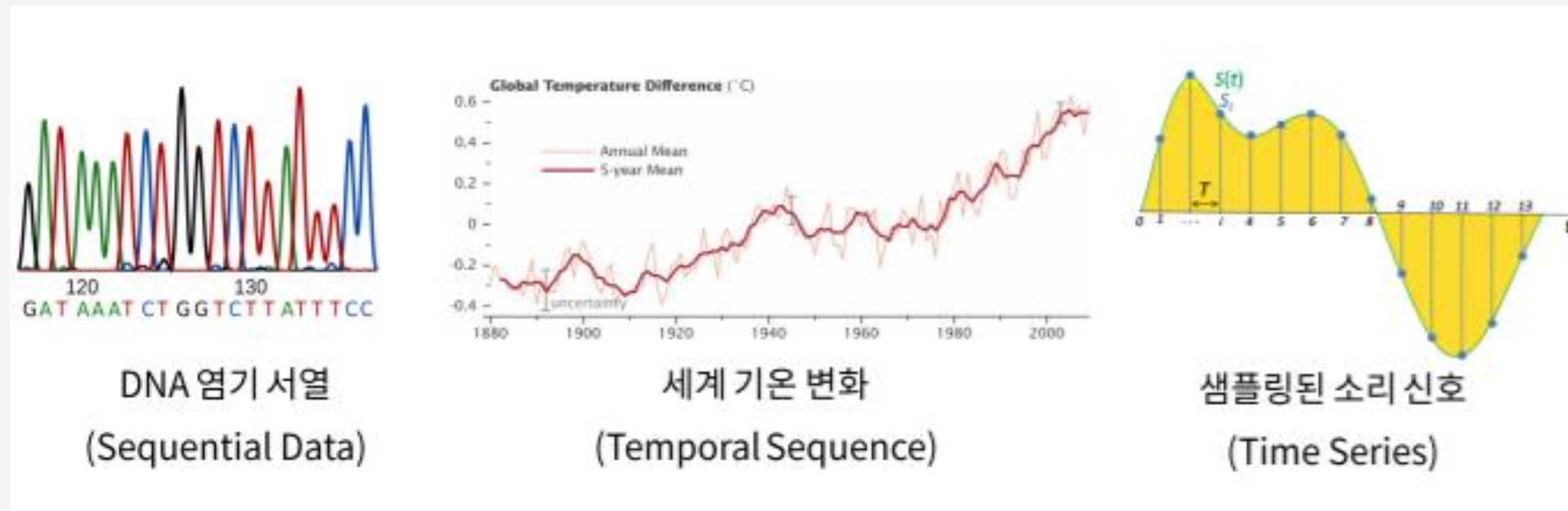


01. 순차데이터란?

순차 데이터 : 순서가 의미가 있으며, 순서가 달라질 경우 의미가 손상되는 데이터

Time series data : 일정한 시간차

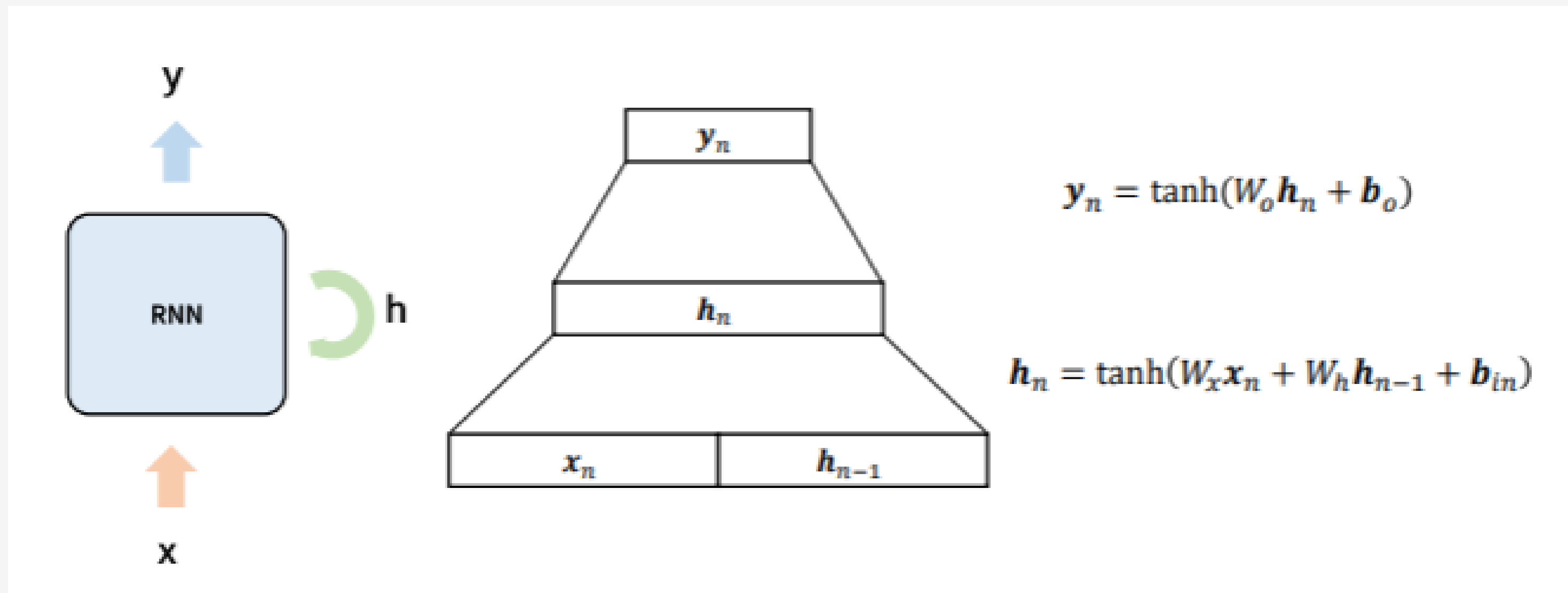
Sequential Data: 순차적 의미가 있는 경우



-> RNN : 순차적인 정보를 가지는 데이터를 효과적으로 학습하기 위해 만들어진 모델

01. RNN이란?

(Valina) RNN(Recurrent Neural Network)



01. RNN이란?



$|X| = (\text{batch_size}, n, \text{input_size})$
 where $X = \{x_1, x_2, \dots, x_n\}$

```
import torch
import numpy as np
```

```
input_size = 4
hidden_size = 2
```

```
# 1-hot encoding
```

```
h = [1, 0, 0, 0] input_size
```

```
e = [0, 1, 0, 0]
```

```
l = [0, 0, 1, 0]
```

```
o = [0, 0, 0, 1]
```

```
input_data_np = np.array([[h, e, l, l, o],
                          [e, o, l, l, l],
                          [l, l, e, e, l]], dtype=np.float32)
```

length_size

3개의 단어를 하나의 batch로 묶음

```
# transform as torch tensor
```

```
input_data = torch.Tensor(input_data_np)
```

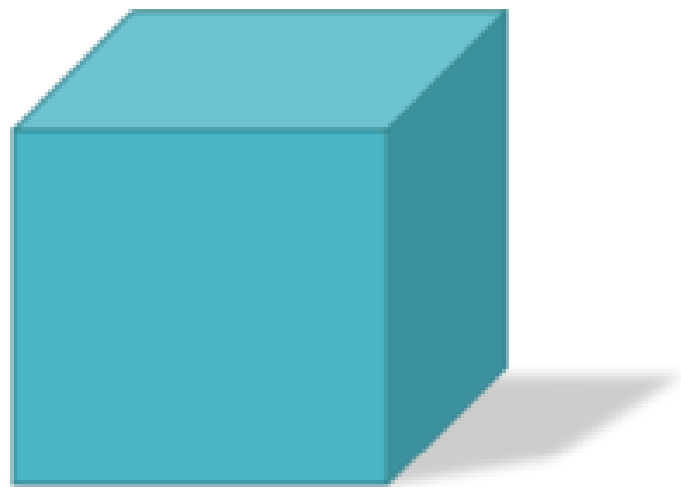
```
rnn = torch.nn.RNN(input_size, hidden_size)
```

```
outputs, _status = rnn(input_data)
```

input_size = 4
 batch_size = 3
 length_size = 5

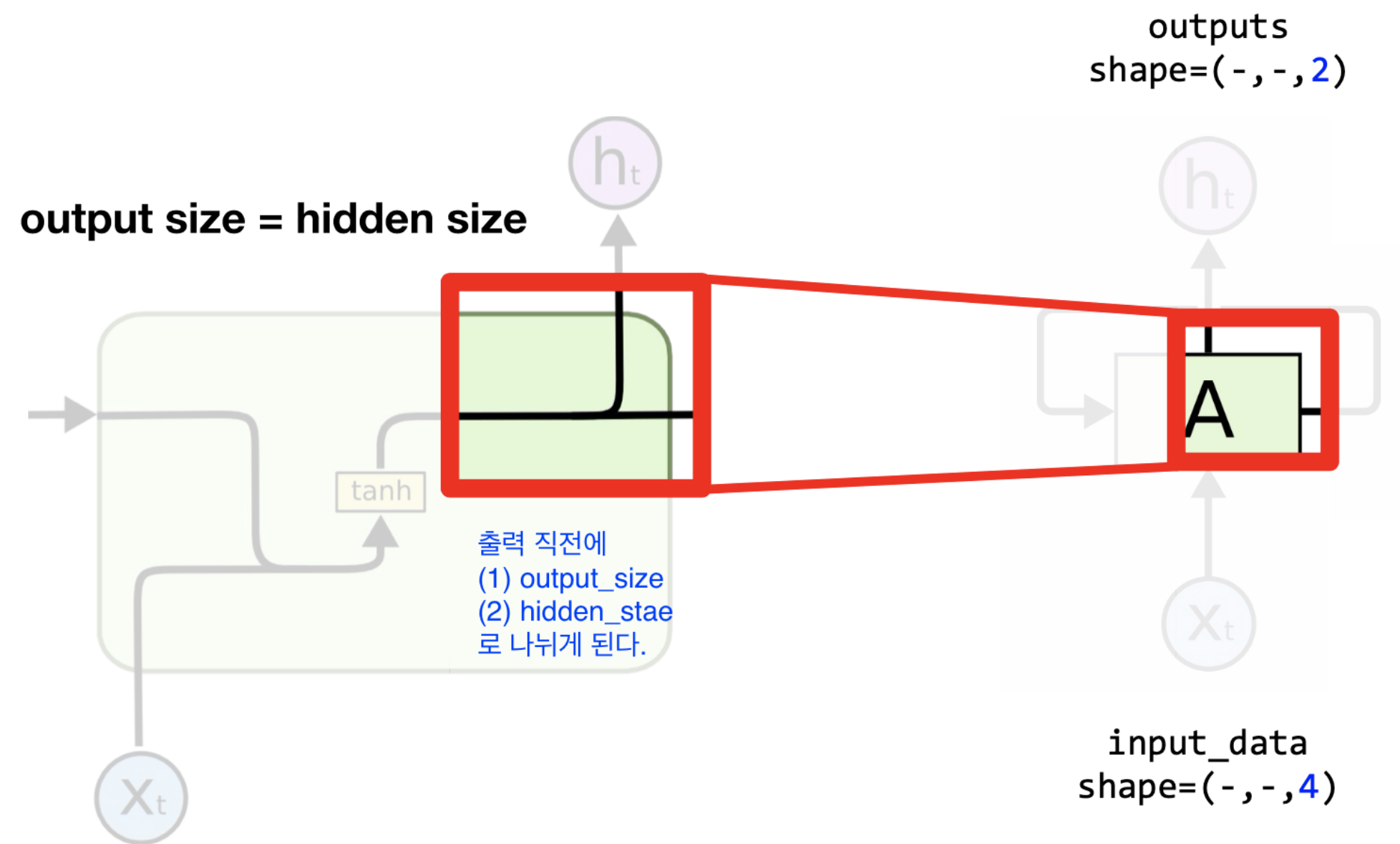
01. RNN이란?

RNN(Recurrent Neural Network)

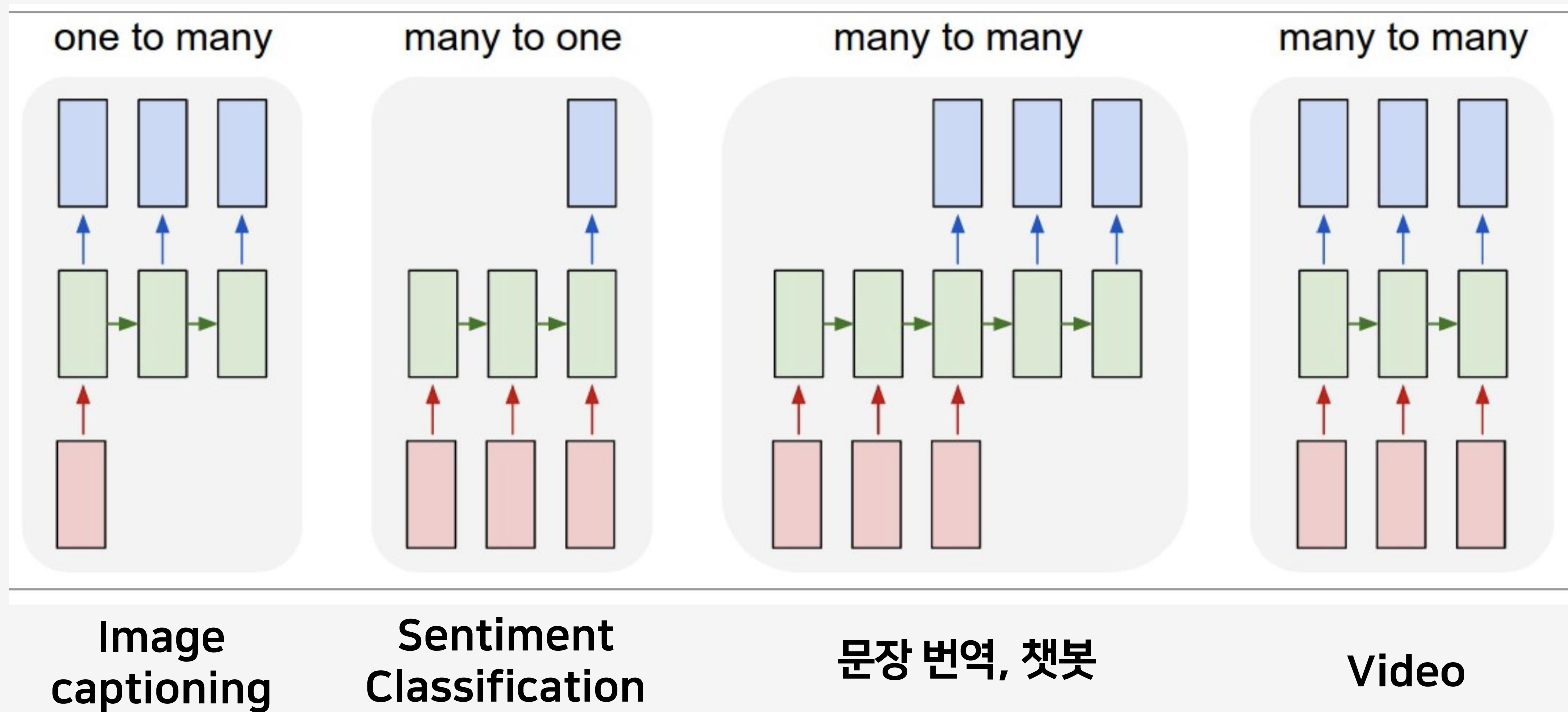


$|h_{1:n}| = (\text{batch_size}, n, \text{hidden_size})$
 where $h_{1:n} = [h_1; h_2; \dots; h_n]$.

↓
 y



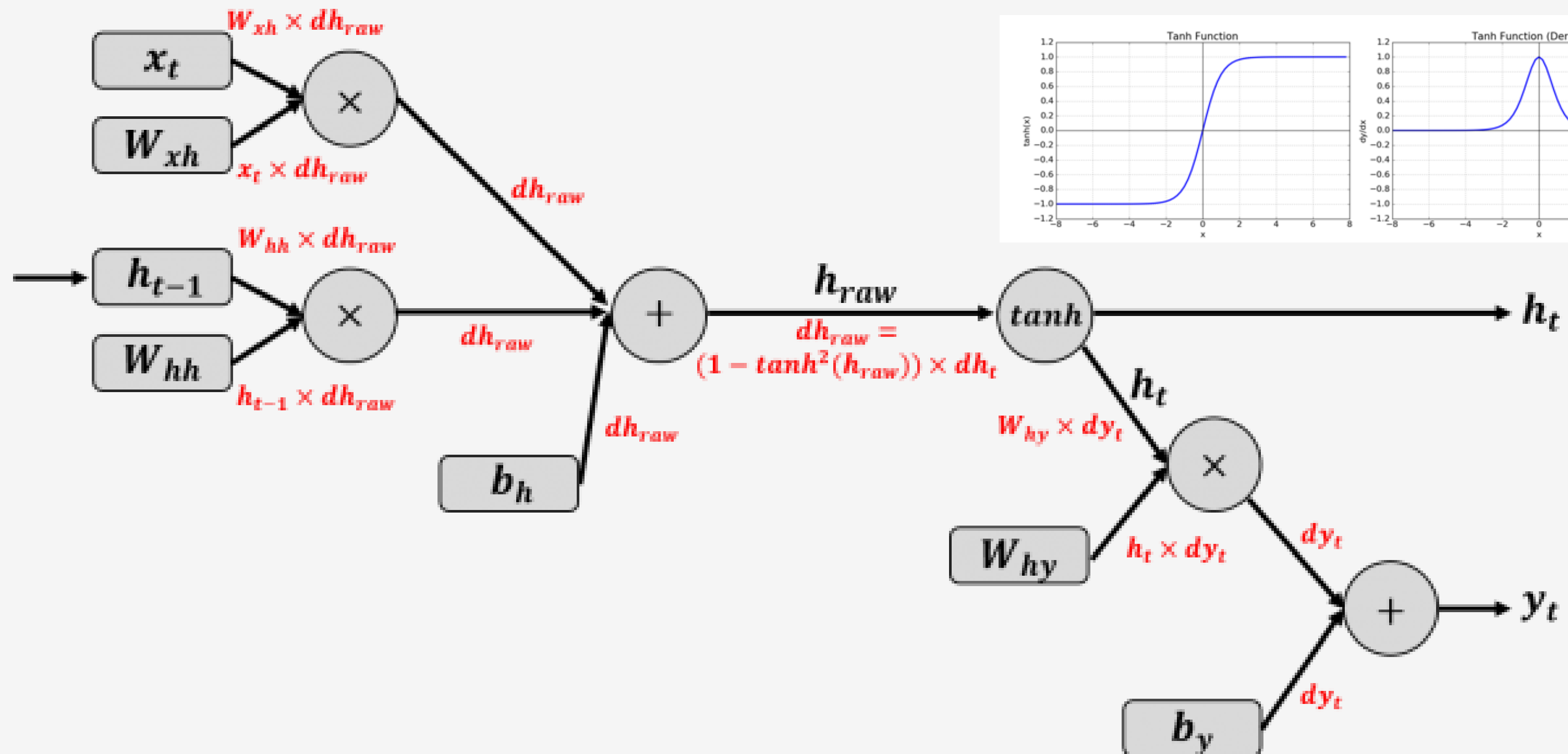
01. RNN이란?



01. RNN이란?

BPTT(Backpropagation Through Time)

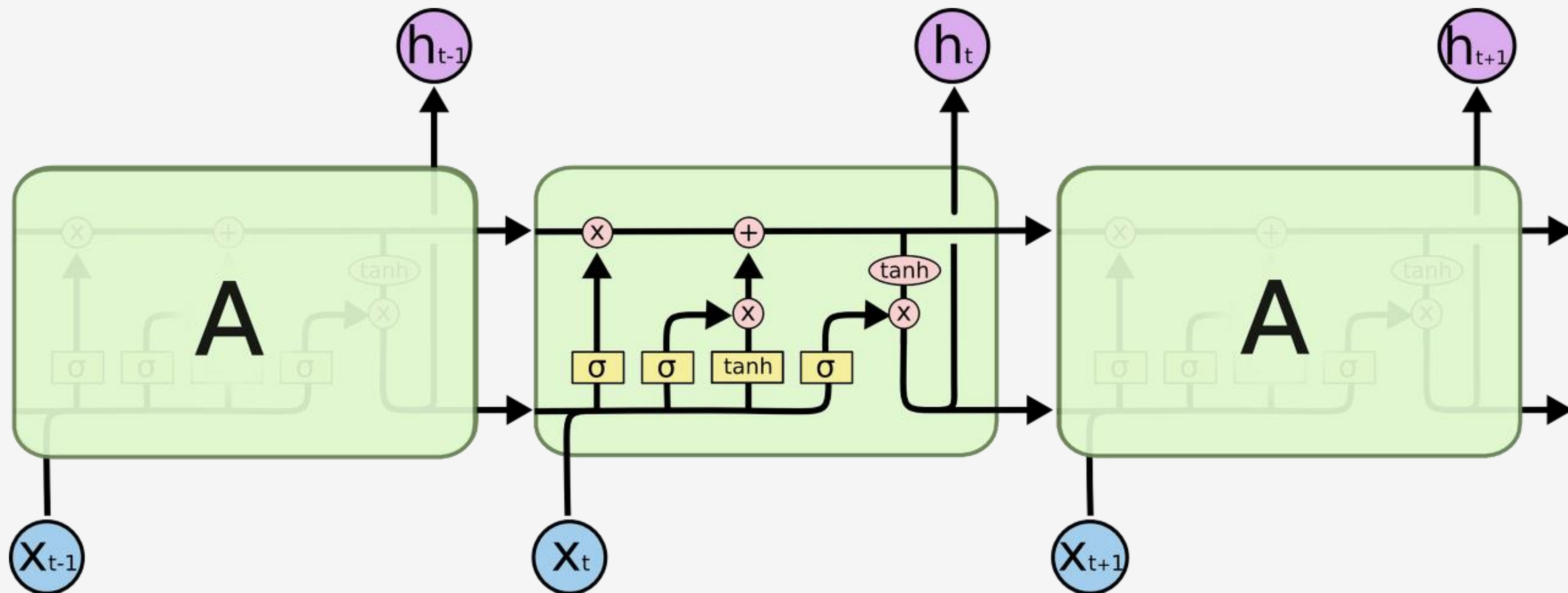
Gradient vanishing 문제 발생 - > Long-term dependencies (장기 의존성 문제)



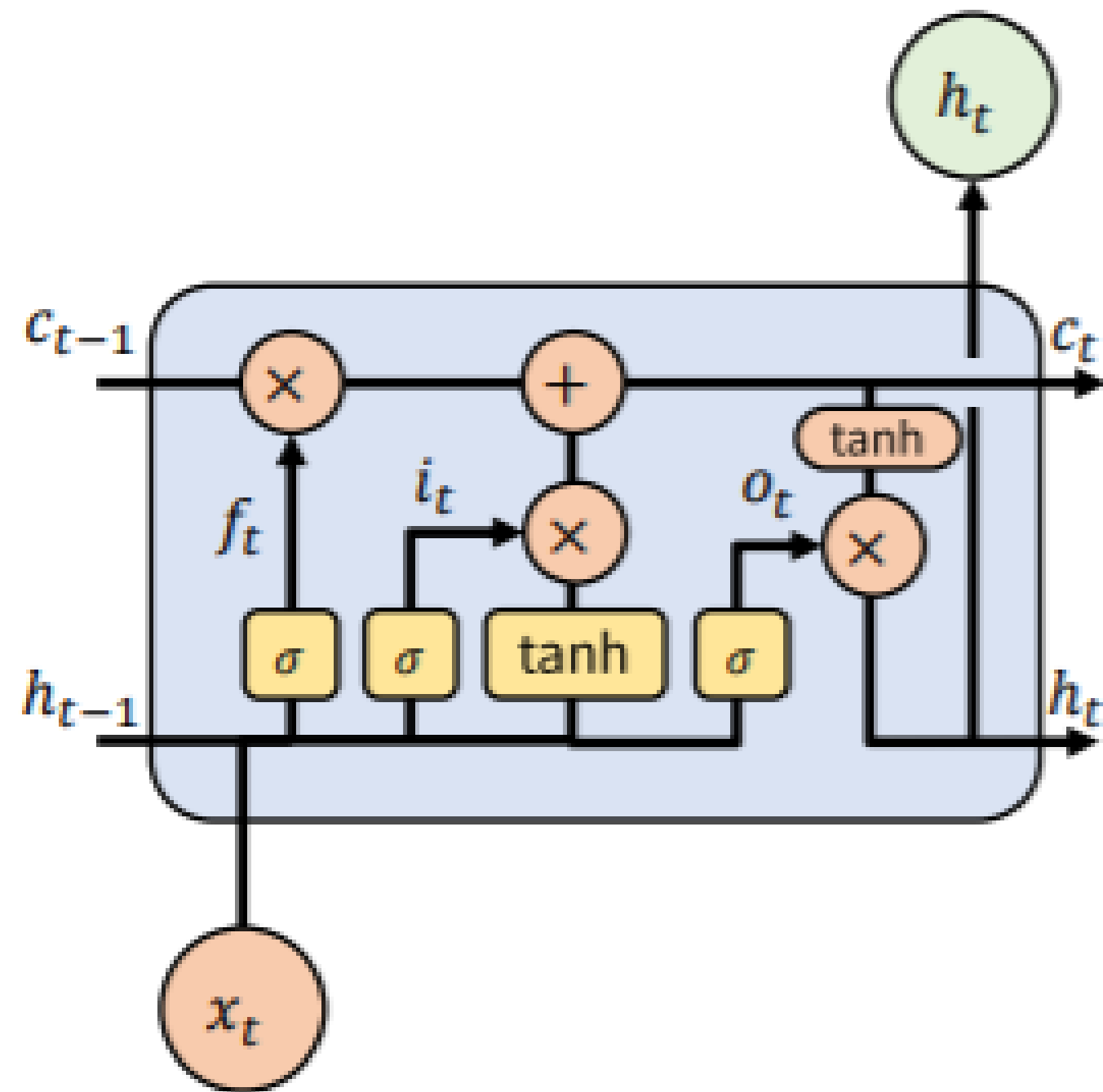
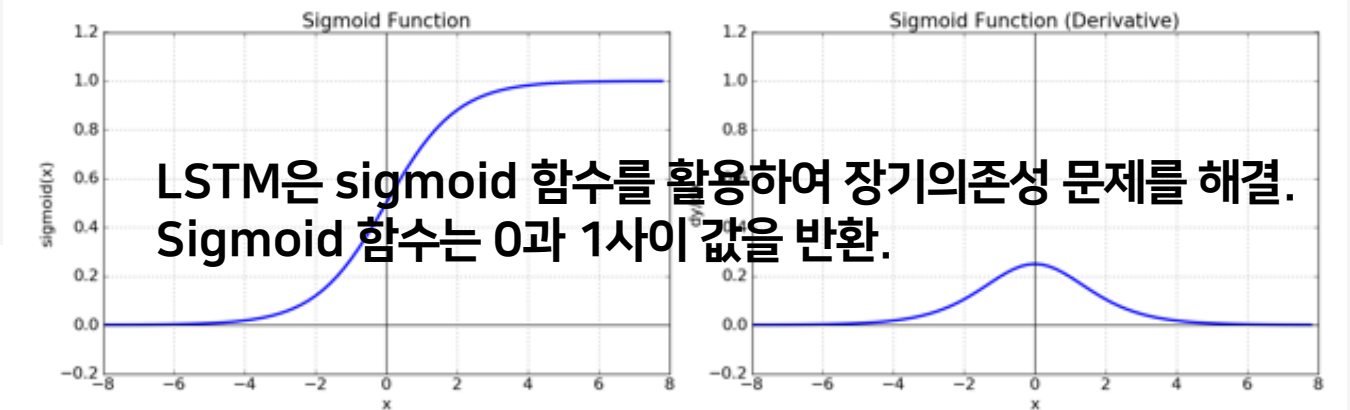
01. LSTM

LSTM (Long Short-Term Memory)

: 기억할 것은 오래 기억하고, 잊을 것은 빨리 잊어버림



01. LSTM



$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \quad \text{Forget gate}$$

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \quad \text{Input gate}$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad \text{Output gate}$$

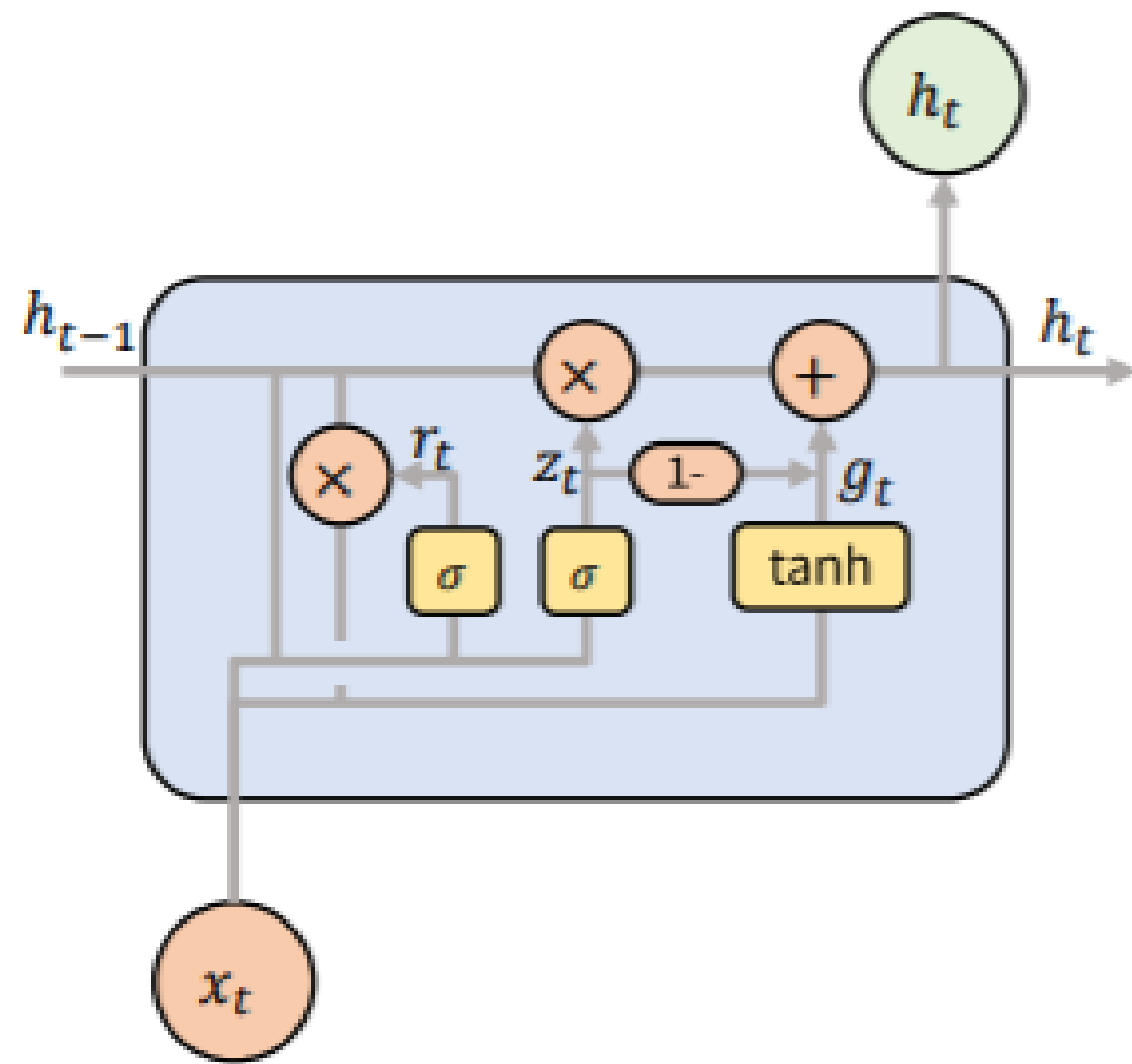
$$g_t = \tanh(W_{xg}x_t + W_{hg}h_{t-1} + b_g) \quad \text{현재 Cell state}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \quad \text{Cell update}$$

$$h_t = o_t \odot \tanh(c_t) \quad \text{Hidden gate}$$

01. GRU

GRU (Gated Recurrent Unit)



$$r_t = \sigma(W_{xr}x_t + W_{hr}h_{t-1} + b_r)$$

$$z_t = \sigma(W_{xz}x_t + W_{hz}h_{t-1} + b_z)$$

$$g_t = \tanh(W_{xg}x_t + W_{hg}(r_t \odot h_{t-1}) + b_g)$$

$$h_t = z_t \odot h_{t-1} + (1 - z_t)g_t$$

- Cell state 없고, hidden state만 존재
- Forget gate와 input gate를 update gate(z)가 제어.
- Reset gate 추가 : 이전 hidden state를 얼마나 사용할지 결정




02. Preprocessing

Cleaning (정제) : 코퍼스로부터 노이즈 데이터 제거

- 길이가 짧은 단어 제거
- 대소문자 통합
- 특수문자 제거 (ex. * ` ~ etc)
- 불용어(stopword) : 큰 의미가 없는 단어 제거

<https://www.ranks.nl/stopwords/korean>

Korean Stopwords

0 |   

[Home](#) > [Resources](#) > [Stopwords](#) > [Korean](#)

Korean Stopwords		
아	어찌됐든	하기보다는
휴	그위에	차라리
아이구	게다가	하는 편이 낫다
아이쿠	점에서 보아	흐흐
아이고	비추어 보아	놀라다
어	고려하면	상대적으로 말하
나	하게될것이다	자면
우리	일것이다	마치
저희	비교적	아니라면
따라	좀	싹
의해	보다더	그렇지 않으면
을	비하면	그렇지 않다면
를	시키다	안 그러면
에	하게하다	아니었다면
의	할만하다	하든지
가	의해서	아니면
으로	연이서	이라면
로	이어서	좋아
에게	잇따라	알았어
뿐이다	뒤따라	하는것도
의거하여	뒤이어	그만이다

02. Preprocessing

Normalization (정규화) : 표현 방법이 다른 단어를 통합시켜서 같은 단어로 만들어준다.

- Stemming (어간 추출)

```
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
s = PorterStemmer()
text="This was not the map we found in Billy Bones's chest, but an accurate copy, complete in all things--names and heights and soundings--with the single exception of the red crosses and the written notes."
```

```
print([s.stem(w) for w in words])
```

```
['thi', 'wa', 'not', 'the', 'map', 'we', 'found', 'in', 'billi', 'bone',
'chest', ',', 'but', 'an', 'accur', 'copi', ',', 'complet', 'in', 'all',
'g', '--', 'name', 'and', 'height', 'and', 'sound', '--', 'with', 'the',
'l', 'except', 'of', 'the', 'red', 'cross', 'and', 'the', 'written', 'no',
'.']
```

- Lemmatization (표제어 추출)

ex) are, am, is -> be

```
from nltk.stem import WordNetLemmatizer
n=WordNetLemmatizer()
```

```
n.lemmatize('dies', 'v')
```

```
'die'
```

02. Preprocessing

Padding : 가변적인 길이를 가지는 문장을 같은 길이로 맞춰주는 방법 (zero-padding)

-> 앞에서부터 채우는 pre-padding, 뒤에서부터 채우는 post-padding으로 나뉘짐.

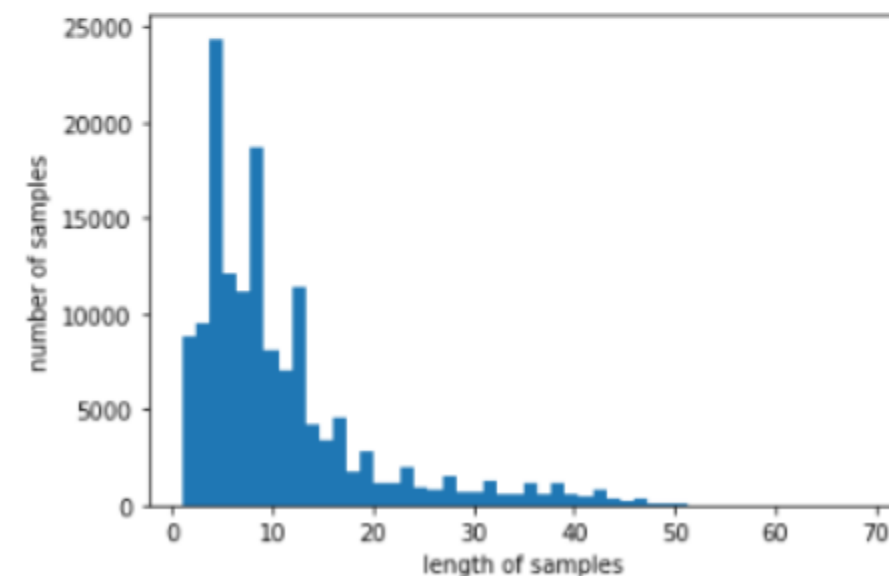
D. M. Reddy et al, "Effects of Padding on LSTMs and CNNs", 2019.

Table 1: LSTM pre-padding vs LSTM post-padding (%)

	LSTM-4 Pre-Padding	LSTM-4 Post-Padding
Train	80.072	49.977
Test	80.321	50.117
Epochs	9	6

```
print('리뷰의 최대 길이 :', max(len(l) for l in X_train))
print('리뷰의 평균 길이 :', sum(map(len, X_train))/len(X_train))
plt.hist([len(s) for s in X_train], bins=50)
plt.xlabel('length of samples')
plt.ylabel('number of samples')
plt.show()
```

리뷰의 최대 길이 : 69
리뷰의 평균 길이 : 10.812485361182679



02. Tokenization

Tokenization(토큰화): 주어진 corpus에서 token 단위로 나누는 작업

Word tokenization , sentence tokenization

```
from nltk.tokenize import word_tokenize
print(word_tokenize("Don't be fooled by the dark sounding name, Mr. Jones Orphanage is as cheery as cheery goes for a pastry shop."))
```

```
['Do', "n't", 'be', 'fooled', 'by', 'the', 'dark', 'sounding', 'name', ',', 'M', 'r.', 'Jones', "'s", 'Orphanage', 'is', 'as', 'cheery', 'as', 'cheery', 'goes', 'for', 'a', 'pastry', 'shop', '.']
```

영어 : NLTK의 word_tokenize, wordPunctokenizer, Keras의 text_to_word_sequence 등..

-> 구두점, 특수 문자(!) 등을 단순 제외해서는 안 됨. Ex) 01/02/06 : 날짜

-> 줄임말, 단어 내 띄어쓰기 ex) we're : we are

02. Tokenization

한국어 : 교착어 , 띄어쓰기 잘 안 이루어짐.

-> KoNLPy 의 형태소 분석기: Okt(Open Korea Text), Mecab, Komoran, Hannanum, Kkma(꼬꼬마)

-> khaii(카카오 발표)

PyKoSpacing : 띄어쓰기 패키지
Py-Hanspell : 맞춤법, 띄어쓰기 보장 패키지

```
from konlpy.tag import Okt
okt=Okt()
print(okt.morphs("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

```
['열심히', '코딩', '한', '당신', ',', '연휴', '에는', '여행', '을', '가봐요']
```

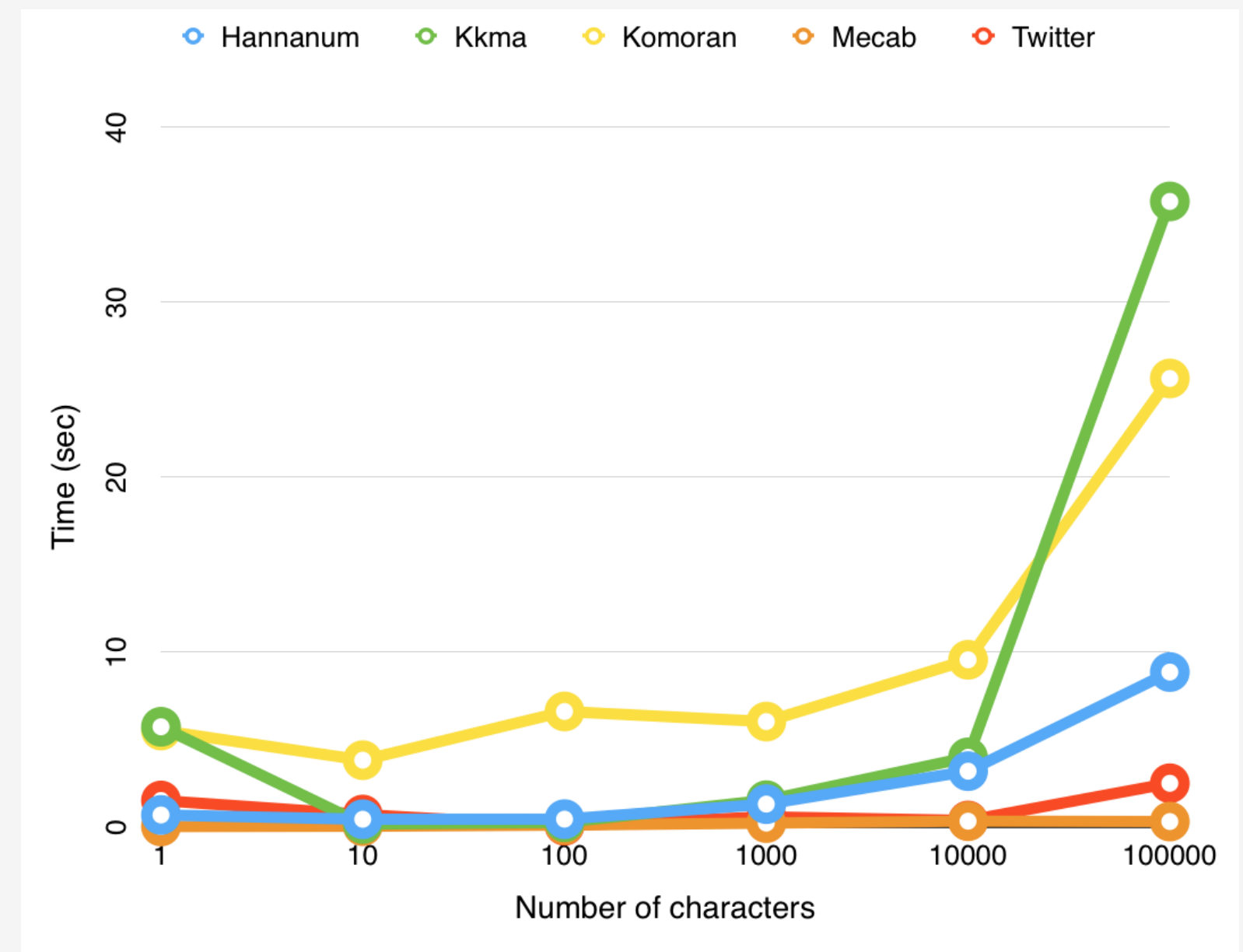
```
print(okt.pos("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

Part-of-speech tagging
: 품사 태깅

```
[('열심히', 'Adverb'), ('코딩', 'Noun'), ('한', 'Josa'), ('당신', 'Noun'), (',', 'Punctuation'), ('연휴', 'Noun'), ('에는', 'Josa'), ('여행', 'Noun'), ('을', 'Josa'), ('가봐요', 'Verb')]
```

```
print(okt.nouns("열심히 코딩한 당신, 연휴에는 여행을 가봐요"))
```

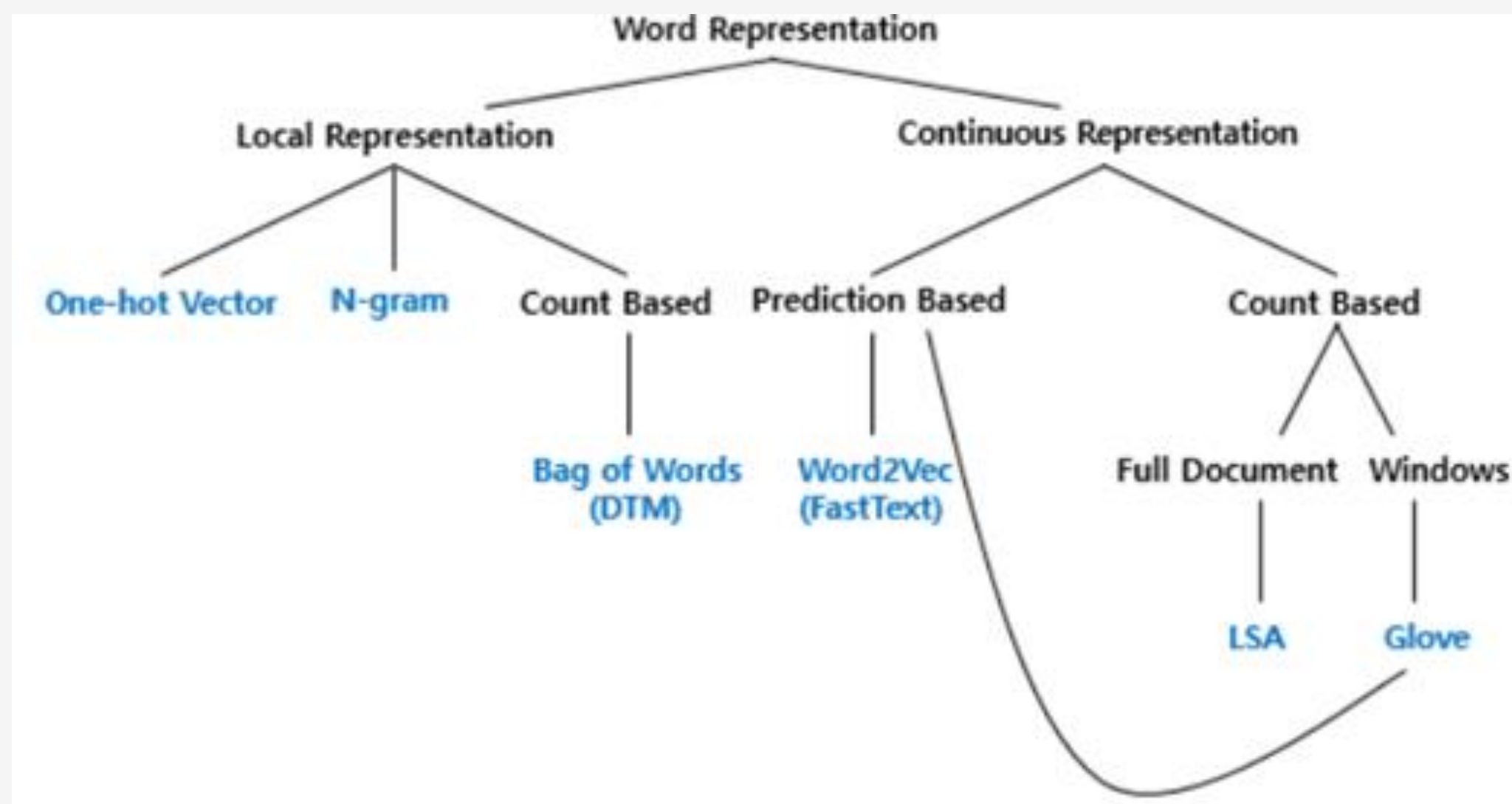
```
['코딩', '당신', '연휴', '여행']
```



02. Token Embedding – Local Representation

다양한 단어의 표현 방법

- 국소(이산) 표현 (Local Representation) : 해당 단어 자체만 보고 단어표현.
- 분산 표현(Distributed Representation): 주변을 참고하여 단어 표현. 뉘앙스 표현 가능



```
sub_text="점심 먹으러 갈래 메뉴는 햄버거 최고야"
encoded=t.texts_to_sequences([sub_text])[0]
print(encoded)
```

```
[2, 5, 1, 6, 3, 7]
```

```
one_hot = to_categorical(encoded)
print(one_hot)
```

```
[[0. 0. 1. 0. 0. 0. 0. 0.] #인덱스 2의 원-핫 벡터
 [0. 0. 0. 0. 0. 1. 0. 0.] #인덱스 5의 원-핫 벡터
 [0. 1. 0. 0. 0. 0. 0. 0.] #인덱스 1의 원-핫 벡터
 [0. 0. 0. 0. 0. 0. 1. 0.] #인덱스 6의 원-핫 벡터
 [0. 0. 0. 1. 0. 0. 0. 0.] #인덱스 3의 원-핫 벡터
 [0. 0. 0. 0. 0. 0. 0. 1.]] #인덱스 7의 원-핫 벡터
```

02. Token Embedding – Local Representation

Bag of Words

: 단어의 등장 순서는 고려하지 않고, 출현 빈도(카운트)를 중심으로 텍스트 데이터를 수치화하는 방법
: 분류 문제, 여러 문서 간의 유사도에 활용.

BoW 만드는 과정

- 1) 각 단어에 고유한 정수 인덱스 부여
- 2) 각 인덱스의 위치에 단어 토큰의 등장 횟수를 기록한 벡터를 생성.

```
from konlpy.tag import Okt
import re
okt=Okt()

token=re.sub("(\\W?)","", "우리 집 강아지는 동생을 좋아할까, 간식을 더 좋아할까?")
token=okt.morphs(token)

word2index={}
bow=[]
for voca in token:
    if voca not in word2index.keys():
        word2index[voca]=len(word2index)
        bow.insert(len(word2index)-1,1)
    else:
        index=word2index.get(voca)
        bow[index]=bow[index]+1

print(word2index)
bow
```

{'우리': 0, '집': 1, '강아지': 2, '는': 3, '동생': 4, '을': 5, '좋아할까': 6, ' ','': 7, '간식': 8, '더': 9}

[1, 1, 1, 1, 1, 2, 2, 1, 1, 1]

02. Token Embedding – Local Representation

BoW (Bag of Words)의 한계점

1) 희소표현

- : 대부분의 값이 0인 표현을 희소 벡터, 희소 행렬
- : 많은 양의 저장 공간, 계산을 위한 리소스를 필요로 함.
- > 텍스트 전처리를 통해 단어 정규화 필요.

2) 단순 빈도수 기반 접근

- Ex) The, a 공통적으로 많더라도, 유사한 문서 아닐 수 있음.
- > TF-IDF 중요한 단어에 대해 가중치를 부여.

TF-IDF(Term Frequency-Inverse Document Frequency)

- : BoW 내 각 단어에 대한 중요도를 가중치로 주는 방법.
- : 모든 문서에서 자주 등장하는 단어는 중요도가 낮고,
특정 문서에서만 자주 등장하는 단어의 중요도가 높다고 본다.

-tf(d,t) : 특정 문서 d에서의 특정 단어 t의 등장횟수

-df(t): 특정 단어 t가 등장한 문서의 수

-idf(d,t): df(t)에 반비례하는 수

$$idf(d, t) = \log\left(\frac{n}{1 + df(t)}\right)$$

02. Token Embedding – Local Representation

```

▶ import pandas as pd
from math import log
docs = [
    '유튜브 접속 끊겼다',
    '유튜브 서버 터졌다',
    '유튜브 강의 못 듣는다',
    '블랙보드 서버 강의 괜찮다'
]
vocab = list(set(w for doc in docs for w in doc.split()))
#set으로 설정해 중복하는 것 하나만 셈. 그 후 리스트로 변환
vocab.sort()

```

```

▶ N = len(docs)
def tf(t, d):
    return d.count(t)
def idf(t):
    df = 0
    for doc in docs:
        df += t in doc
    return log(N/(df + 1))
def tfidf(t, d):
    return tf(t,d)* idf(t)

```

```

▶ result = []
for i in range(N):
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):
        t = vocab[j]
        result[-1].append(tf(t, d))
# 하나씩 각 docs에서 특정 단어의 출현 횟수를 result에 append
tf_ = pd.DataFrame(result, columns = vocab)
tf_

```

3]:

	강의	괜찮다	끊겼다	듣는다	못	블랙보드	서버	유튜브	접속	터졌다
0	0	0	1	0	0	0	0	1	1	0
1	0	0	0	0	0	0	1	1	0	1
2	1	0	0	1	1	0	0	1	0	0
3	1	1	0	0	0	1	1	0	0	0

02. Token Embedding – Local Representation

```

▶ result = []
for j in range(len(vocab)):
    t = vocab[j]
    result.append(idf(t))
idf_ = pd.DataFrame(result, index = vocab, columns = ["IDF"])
idf_
# log(N/(df + 1))을 반환하는 idf함수를 활용하여, result에 append

```

]:

	IDF
강의	0.287682
괜찮다	0.693147
끓었다	0.693147
듣는다	0.693147
못	0.693147
블랙보드	0.693147
서버	0.287682
유튜브	0.000000
접속	0.693147
터졌다	0.693147

```

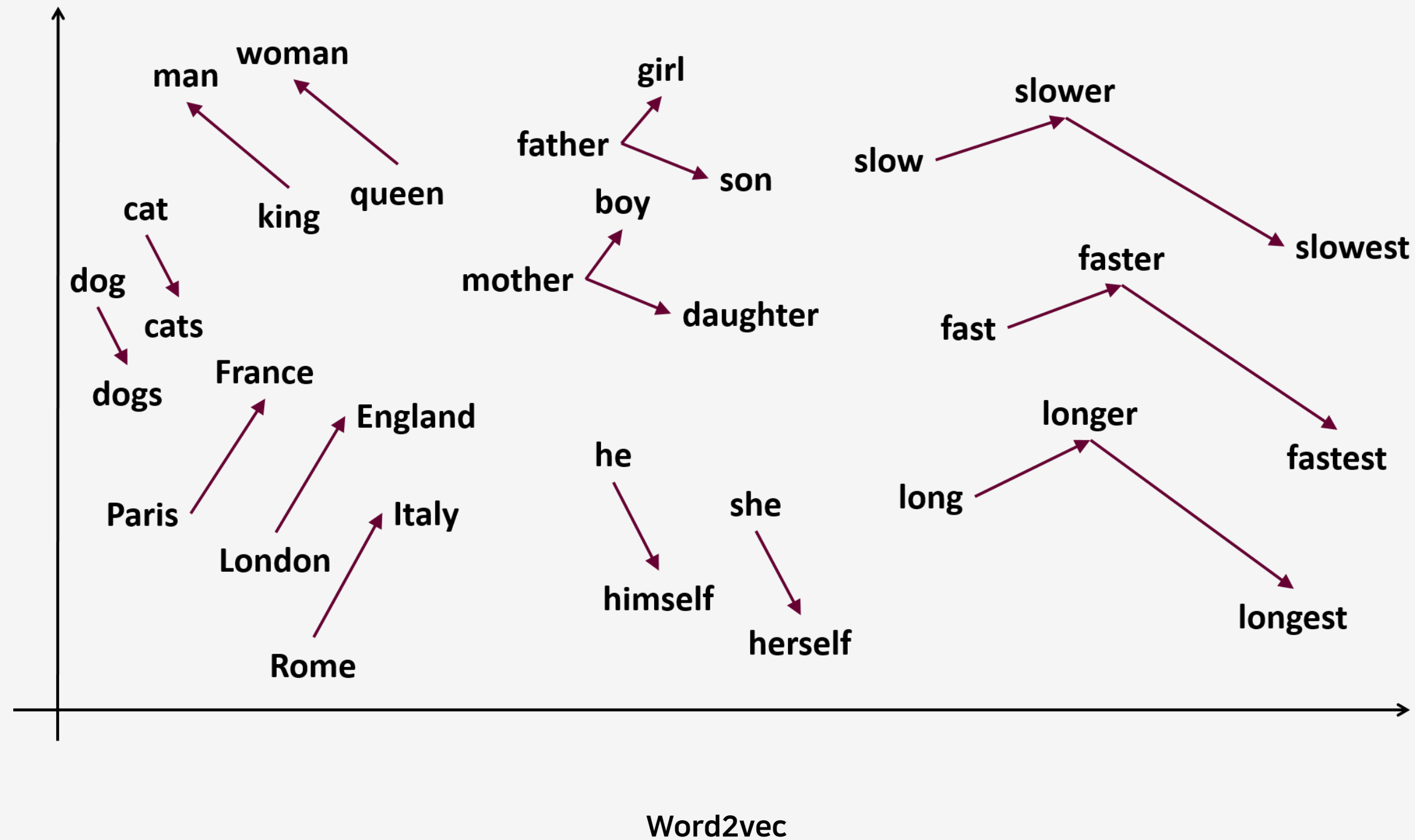
In [35]: ▶ result = []
for i in range(N):
    result.append([])
    d = docs[i]
    for j in range(len(vocab)):
        t = vocab[j]
        result[-1].append(tfidf(t,d))
tfidf_ = pd.DataFrame(result, columns = vocab)
tfidf_

```

Out[35]:

	강의	괜찮다	끓었다	듣는다	못	블랙보드	서버	유튜브	접속	터졌다
0	0.000000	0.000000	0.693147	0.000000	0.000000	0.000000	0.000000	0.0	0.693147	0.000000
1	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.287682	0.0	0.000000	0.693147
2	0.287682	0.000000	0.000000	0.693147	0.693147	0.000000	0.000000	0.0	0.000000	0.000000
3	0.287682	0.693147	0.000000	0.000000	0.000000	0.693147	0.287682	0.0	0.000000	0.000000

02. Token Embedding



02. Document Embedding

```
from tensorflow.keras.layers import Embedding, Dense, LSTM
from tensorflow.keras.models import Sequential
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint
```

임베딩 벡터의 차원은 100으로 정했고, 리뷰 분류를 위해서 LSTM을 사용합니다.

```
model = Sequential()
model.add(Embedding(vocab_size, 100))
model.add(LSTM(128))
model.add(Dense(1, activation='sigmoid'))
```

검증 데이터 손실(val_loss)이 증가하면, 과적합 징후므로 검증 데이터 손실이 4회 증가하면 학습을 조기 종료(EarlyStopping)한다. ModelCheckpoint를 사용하여 검증 데이터의 정확도(val_acc)가 이전보다 좋아질 경우에만 모델을 저장한다.

```
es = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=4)
mc = ModelCheckpoint('best_model.h5', monitor='val_acc', mode='max', verbose=1, save_best_only=True)
```

```
model.compile(optimizer='rmsprop', loss='binary_crossentropy', metrics=['acc'])
history = model.fit(X_train, y_train, epochs=15, callbacks=[es, mc], batch_size=60, validation_split=0.2)
```

저자의 경우 조기 종료 조건에 따라서 9번째 에포크에서 훈련이 멈췄습니다. 훈련이 다 되었다면 이제 테스트 데이터에 대해서 정확도를 측정할 차례입니다. 훈련 과정에서 검증 데이터의 정확도가 가장 높았을 때 저장된 모델인 'best_model.h5'를 로드합니다.

```
loaded_model = load_model('best_model.h5')
print("\n 테스트 정확도: %.4f" % (loaded_model.evaluate(X_test, y_test)[1]))
```

테스트 정확도: 0.8544

```
def sentiment_predict(new_sentence):
    new_sentence = okt.morphs(new_sentence, stem=True) # 토큰화
    new_sentence = [word for word in new_sentence if not word in stopwords] # 불용어 제거
    encoded = tokenizer.texts_to_sequences([new_sentence]) # 정수 인코딩
    pad_new = pad_sequences(encoded, maxlen = max_len) # 패딩
    score = float(loaded_model.predict(pad_new)) # 예측
    if(score > 0.5):
        print("{:.2f}% 확률로 긍정 리뷰입니다.\n".format(score * 100))
    else:
        print("{:.2f}% 확률로 부정 리뷰입니다.\n".format((1 - score) * 100))
```

```
sentiment_predict('이 영화 개꿀잼 ㅋㅋㅋ')
```

97.76% 확률로 긍정 리뷰입니다.

THANK YOU

감사합니다.