SAI

김진명, 문이선,손상원,
이소민,조용재, 박지인

목차

자연어처리 전처리

토큰화 단어집합생성 정수인코딩 패딩 벡터화

START RECENT

- 한글, 공백 제외 모두 제거
- 불용어 제거 (불용어? : 큰 의미가 없는 단어 토큰. 자주 등장하지만 분석을 하는 것에 있어서는 큰 도움이 되지 않는 단어들. 예를 들어 조사가 있다. 은, 는, 이, 가 같이 자주 등장하지만 분석에는 전혀 도움이 되지 않는 것들!)

토큰화

텍스트를 단어 또는 문자 단위로 자르는 것.

- 띄어쓰기
- 형태소 — Mecab 사용

kor_text = "사과의 놀라운 효능이라는 글을 봤어. 그래서 오늘 사과를 먹으려고 했는데 사과가 썩어서 슈퍼에 가서 사과랑 오렌지 사왔어"

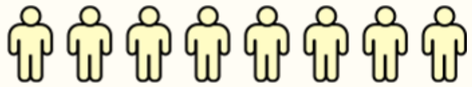
띄어쓰기 = ['사과의', '놀라운', '효능이라는', '글을', '봤어.', '그래서', '오늘', '사과를', '먹으려고', '했는데', '사과가', '썩어서', '슈퍼에', '가서', '사과랑', '오렌지', '사왔어']

형태소 = ['사과', '의', '놀라운', '효능', '이', '라는', '글', '을', '봤', '어', '.', '그래서', '오늘', '사과', '를', '먹', '으려고', '했', '는데', '사과', '가', '썩', '어서', '슈퍼', '에', '가', '서', '사과', '랑', '오렌지', '사', '왔', '어']

단어집합생성

단어 집합(vocabuary)이란 중복을 제거한 텍스트의 총 단어의 집합(set)을 의미

['사과', '의', '놀라운', '효능', '이', '라는', '글', '을', '봤', '어', '.', '그래서', '오늘', '사과', '를', '먹', '으려고', '했', '는데', '사과', '가', '썩', '어서', '슈퍼', '에', '가', '서', '사과', '랑', '오렌지', '사', '왔', '어']



정수인코딩

단어를 빈도수 순으로 정렬한 단어 집합(vocabulary)을 만들고, 빈도수가 높은 순서대로 차례로 낮은 숫자부터 정수를 부여하는 방법

[['어릴', '때', '보', '고', '지금', '다시', '봐도', '재밌', '어요', 'ㅋㅋ'], ['디자인', '을', '배우', '학생', '외국', '디자이너', '그', '일군', '전통', '을', '통해', '발전', '해', '문화', '산업', '부러웠', '는데', '사실', '우리', '나라', '에서', '그', '어려운', '시절', '끝', '까지', '열정', '을', '지킨', '노라노', '같', '전통', '있', '어', '저', '같', '사람', '꿈', '을', '꾸', '고', '이뤄나갈', '수', '있', '다는', '것', '감사', '합니다'], ['폴리스', '스토리', '시리즈', '부터', '뉴', '까지', '버릴', '께', '하나', '없', '음', '최고'], ['연기', '진짜', '개', '쩔', '구나', '지루', '할거', '라고', '생각', '했', '는데', '몰입', '해서', '봤', '다', '그래', '이런', '게', '진짜', '영화', '지'], ['안개', '자욱', '밤하늘', '떠', '있', '초승달', '같', '영화'], ['사랑', '을', '해', '본', '사람', '라면', '처음', '부터', '끝', '까지', '웃', '을', '수', '있', '영화'], ['완전', '감동', '입니다', '다시', '봐도', '감동'], ['개', '전쟁', '나오', '나요', '나오', '면', '빠', '로', '보', '고', '싶', '음'], ['굿'], ['바보', '아니', '라', '병', '쉰', '인', '듯']]

[[78, 27, 9, 4, 50, 41, 79, 16, 28, 29], [188, 5, 80, 189, 190, 191, 42, 192, 114, 5, 193, 194, 21, 115, 195, 196, 13, 51, 81, 116, 30, 42, 197, 117, 118, 31, 198, 5, 199, 200, 17, 114, 7, 82, 52, 17, 43, 201, 5, 202, 4, 203, 14, 7, 83, 32, 204, 84], [205, 119, 206, 53, 207, 31, 208, 209, 54, 10, 25, 11], [44, 33, 120, 210, 211, 212, 213, 68, 45, 34, 13, 214, 121, 15, 2, 215, 69, 8, 33, 3, 35], [216, 217, 218, 219, 7, 220, 17, 3], [122, 5, 21, 36, 43, 123, 124, 53, 118, 31, 85, 5, 14, 7, 3], [125, 37, 221, 41, 79, 37], [120, 222, 55, 223, 55, 86, 224, 46, 9, 4, 47, 25], [56], [225, 87, 88, 226, 227, 57, 89]]

패딩

[[78, 27, 9, 4, 50, 41, 79, 16, 28, 29, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1],

[188, 5, 80, 189, 190, 191, 42, 192, 114, 5, 193, 194, 21, 115, 195, 196, 13, 51, 81, 116, 30, 42, 197, 117, 118, 31,
198, 5, 199, 200, 17, 114, 7, 82, 52, 17, 43, 201, 5, 202, 4, 203, 14, 7, 83, 32, 204, 84, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1],

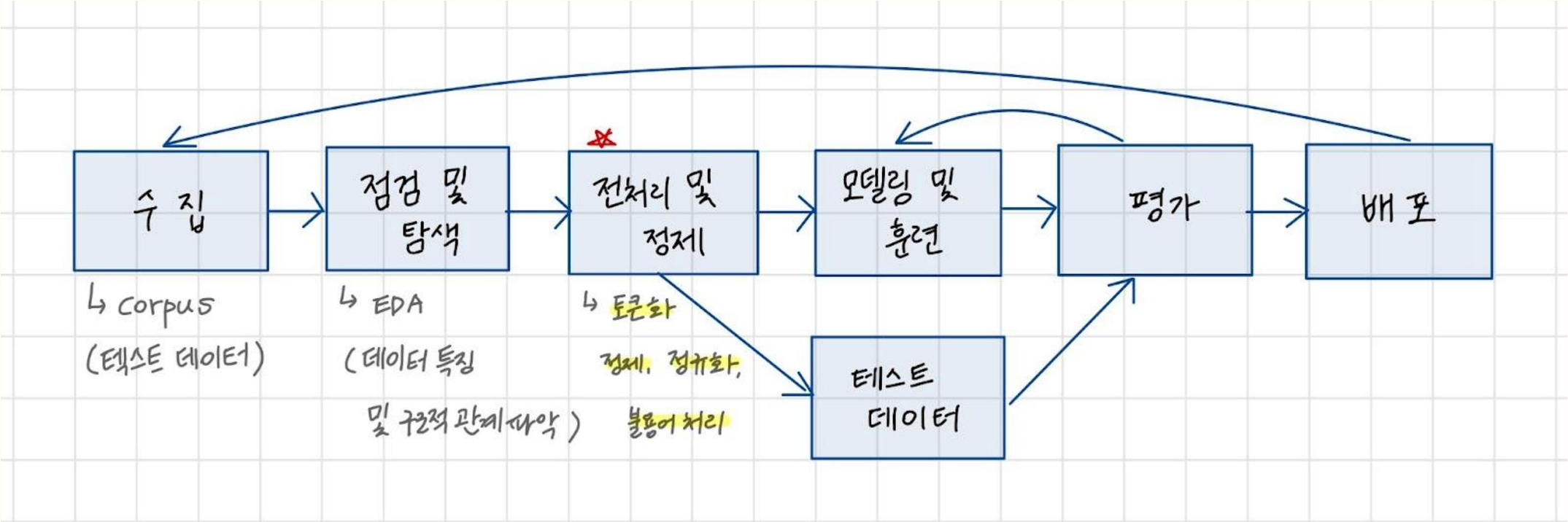
[205, 119, 206, 53, 207, 31, 208, 209, 54, 10, 25, 11, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]

10l pad!

길이가 다른 문장들을 모두 동일한 길이로 바꿔주는 것. 정해준 길이로 모든 샘플들의 길이를 맞춰주되, 길이가 정해준 길이보다 짧은 샘플들에는 'pad' 토큰을 추가하여 길이를 맞춰주는 작업

영어 NLP

Overall Structure



Text Preprocessing : Corpus -> Tokenization & Cleaning & Normalization

Tokenization

단어 토큰화 : 토큰의 단위를 단어로

- 영어 문장 - 띄어쓰기 단위로 잘라도 단어 토큰이 구분됨

  but, **구두점 & 특수문자를 전부 제거하면 토큰이 의미를 잃어버릴 수도 있음**

  (nltk의 word.tokenize & WordPuncTokenizer)

- 토큰화 고려사항

  1. 구두점이나 특수문자 단순제거 금지

     ex) 마침표 : 문장의 경계를 나타내기도 하지만, 특정 직업이나 수치 등을 나타냄

  2. 줄임말과 단어 내에 띄어쓰기가 있는 경우

     ex) We're ⇒ We are의 줄임말, New York ⇒ 한 단어

  3. 표준 토큰화 Penn Treebank Tokenization 규칙

     a. 하이푼으로 구성된 단어는 하나로

     b. doesn't와 같은 접어는 분리

문장 토큰화 : 토큰의 단위가 문장일 때

- 코퍼스가 정제되지 않았을 때 문장 토큰화가 필요할 수 있음
- 마침표 or 느낌표와 같이 **보통 문장의 끝을 나타내는 특수문자 기준으로 자르면 X**

  (nltk의 sent_tokenize)

  → 예외 사항을 발생시키는 마침표 처리를 위해 **이진분류기**를 사용하기도 함.

  class1) 마침표 단어의 일부분일 경우

  class2) 마침표가 문장의 구분자일 경우

  ⇒ 머신러닝으로 학습

- 영어 NLP : 합성어(ex. New York)나 줄임말(ex. he's)에 대한 예외처리만 해도, 띄어쓰기를 기준으로 하는 토큰화 수행 시 잘 작동.

품사 태깅 : 단어는 같지만 품사에 따라 단어의 의미가 달라지는 경우

- NLTK : 영어 품사 태깅

  (nltk.tag의 pos_tag)

- 중요도 : 한국어 NLP >> 영어 NLP

Cleaning & Normalization

- 토큰화 작업 전/후에는 텍스트 용도에 맞게 정제 및 정규화 과정 필요

정제 : copus로부터 Noise data 제거

정규화 : 표현 방법이 다른 단어들을 통합해서 같은 단어로 만들기

1. **규칙에 기반한 표기가 다른 단어들의 통합**

   ⇒ 어간 추출(stemming) & 표제어 추출(lemmatization)

   ex) US = USA, uh-huh = uhhuh

2. **대, 소문자 통합**

   - 무조건 다 통합 X

     ex) US ≠ us

   - 회사나 사람 이름은 대문자 유지하는 것이 좋음

3. **불필요한 단어 제거(noise data)**

   ⇒ 불용어 제거, 빈도수가 적거나 길이가 짧은 단어 제거

   1. 등장 빈도수가 적은 단어
   2. 길이가 짧은 단어

   cf) 영어에서는 대부분 짧은 단어들이 불용어

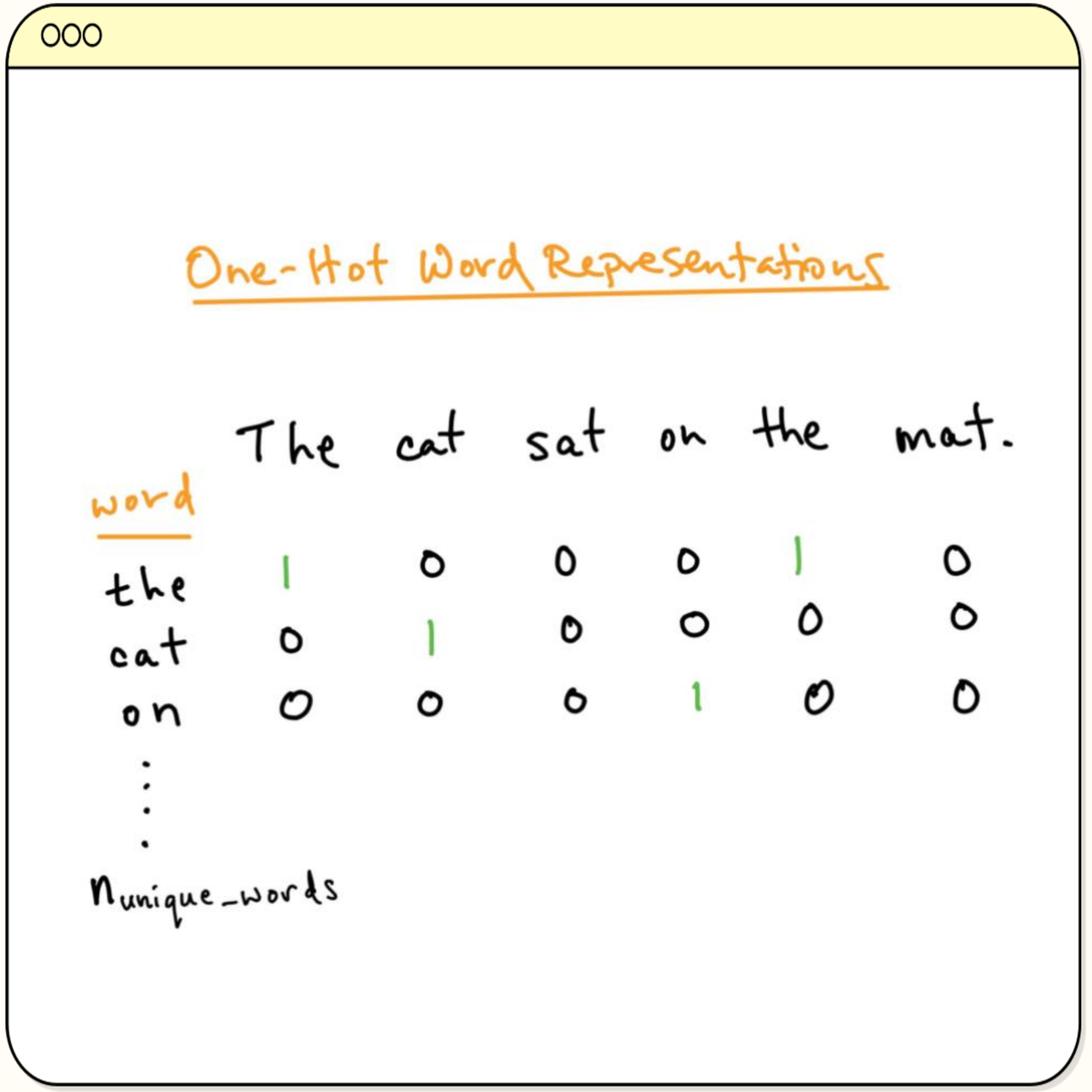
   정규 표현식 : corpus 내에 계속해서 등장하는 글자들을 규칙에 기반하여 한번에 제거하는 과정

임베딩 이전의 전처리

기본적으로 자연어는 비정형 데이터이기 때문에, 모델을 통한 학습을 진행하기 위해 데이터 전처리가 필요

기존에는 주요 단어들을 토큰으로 만들어 낸 후, 이 토큰들에 원핫 인코딩을 해서 학습 진행

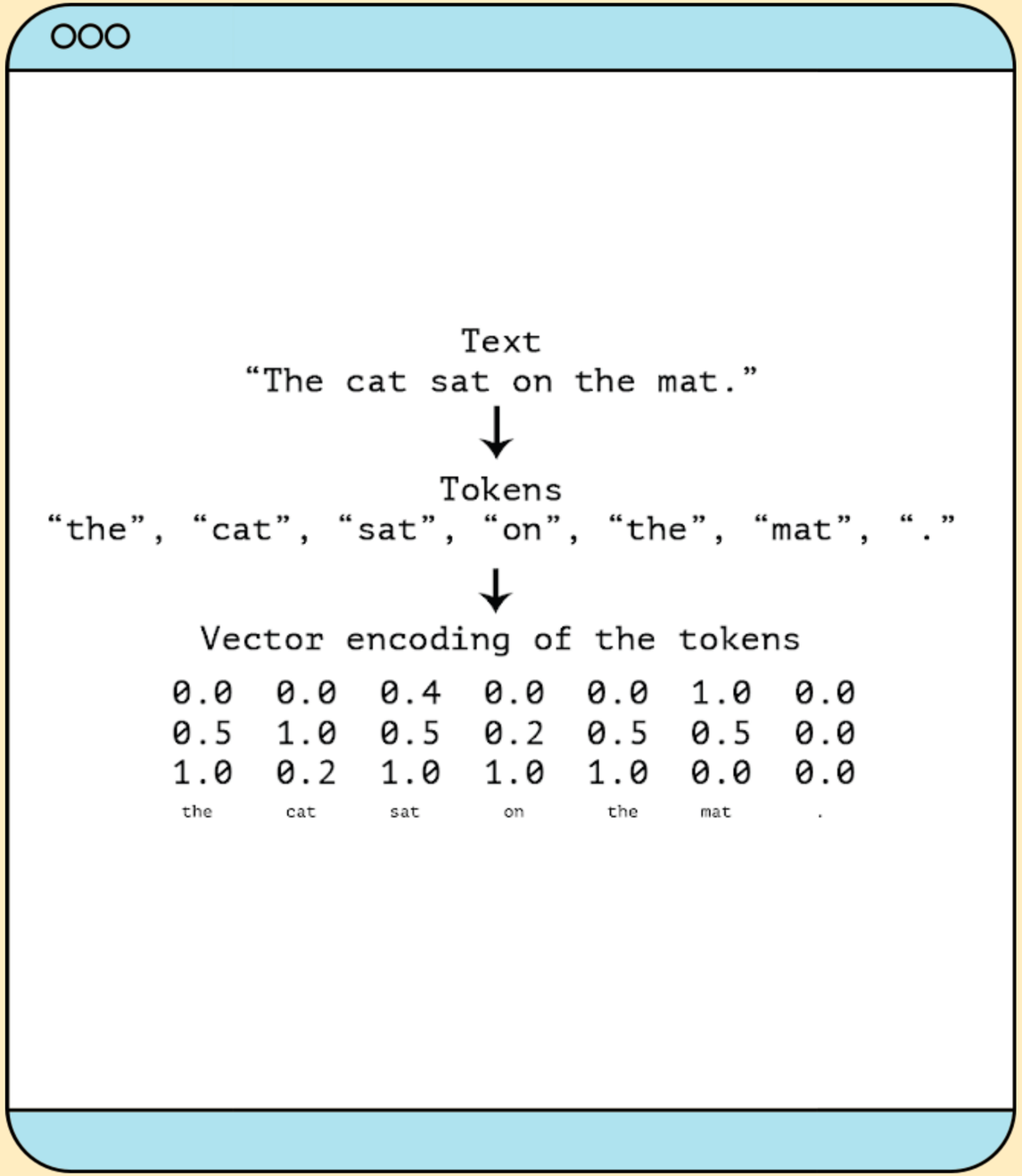
하지만, 1번 이상 존재하는 단어가 많아질 수록 원핫 인코딩 행렬이 방대하게 커지는 등 다양한 문제점이 있어서 학습에 악영향을 미침



워드 임베딩

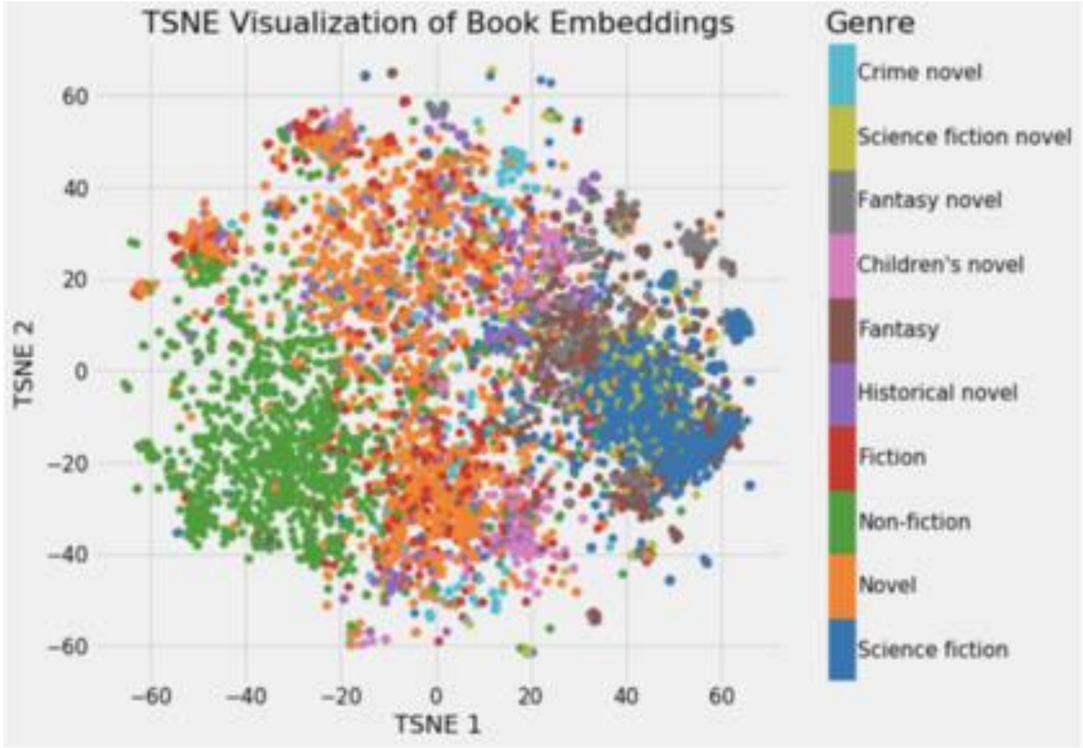
그래서 기존 방식을 대신에 단어들을 “벡터화” 시킴. 즉, 각 단어들이 수치적 방향성을 갖도록 만드는 방식이 word embedding

기존에는 0 혹은 1의 값을 갖지만, embedding 처리를 진행한 후엔 실수값을 가진 벡터가 된다.



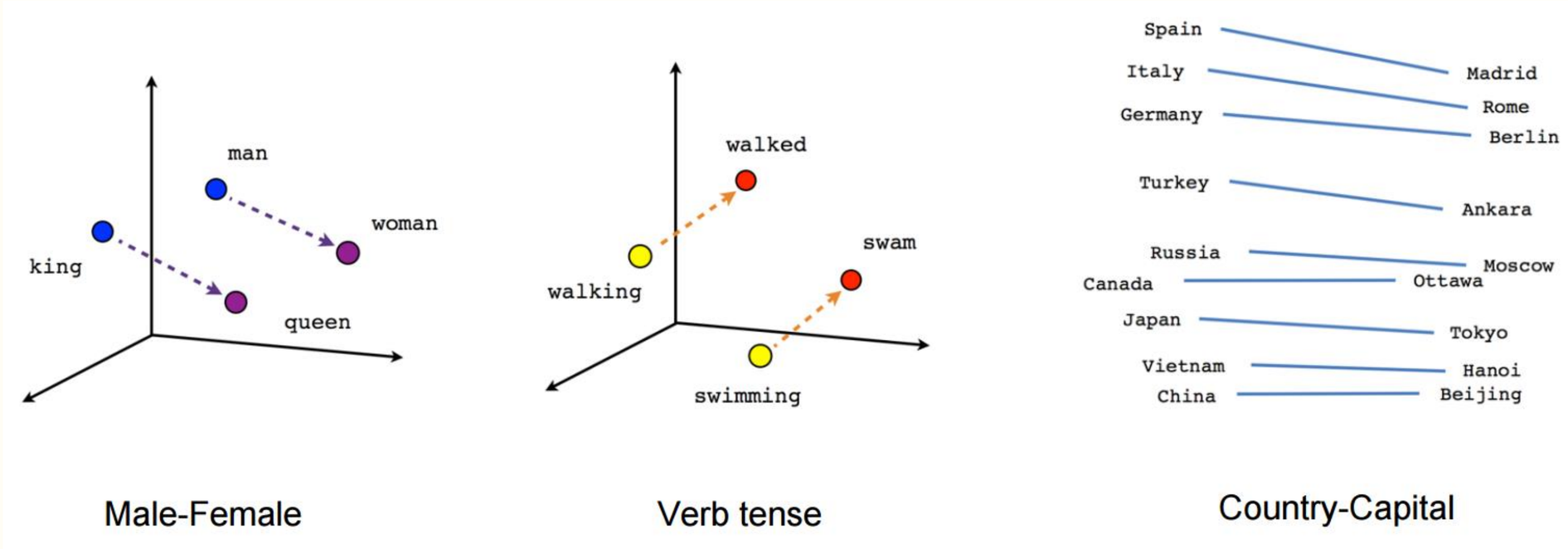
워드 임베딩의 특징(1)

같은 그룹 내 단어들은 비슷한 방향성을 갖는다.



Embeddings Colored by Genre

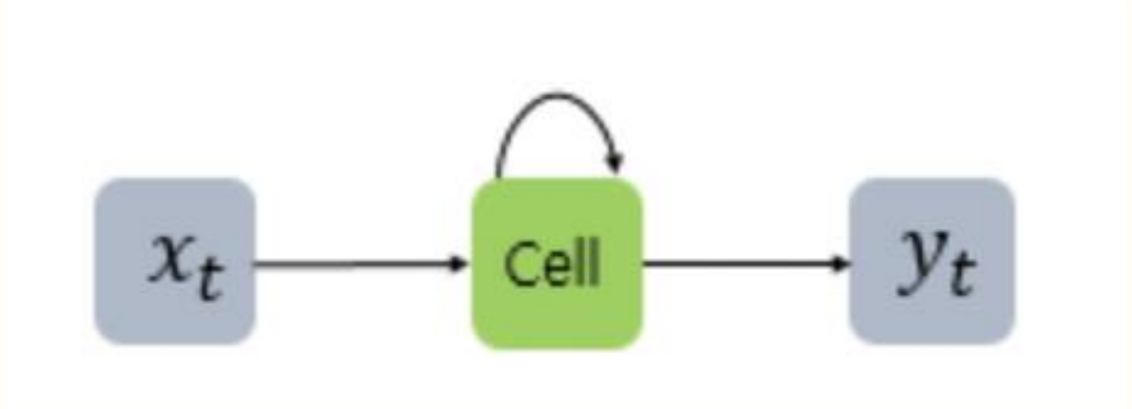
워드 임베딩



각 임베딩 벡터의 방향이 의미 가치를 담고 있기 때문에, 관련된 단어간 벡터의 차가 동일하게 설정된다.

ex) (man의 벡터) - (woman의 벡터) = (king의 벡터) - (queen의 벡터)

RNN



입력과 출력을 시퀀스 단위로 처리하는 모델

은닉층의 노드에서 활성화 함수를 통해 나온 결과값 출력층 방향 + 다시 은닉층 노드의 다음 계산의 입력으로 보냄

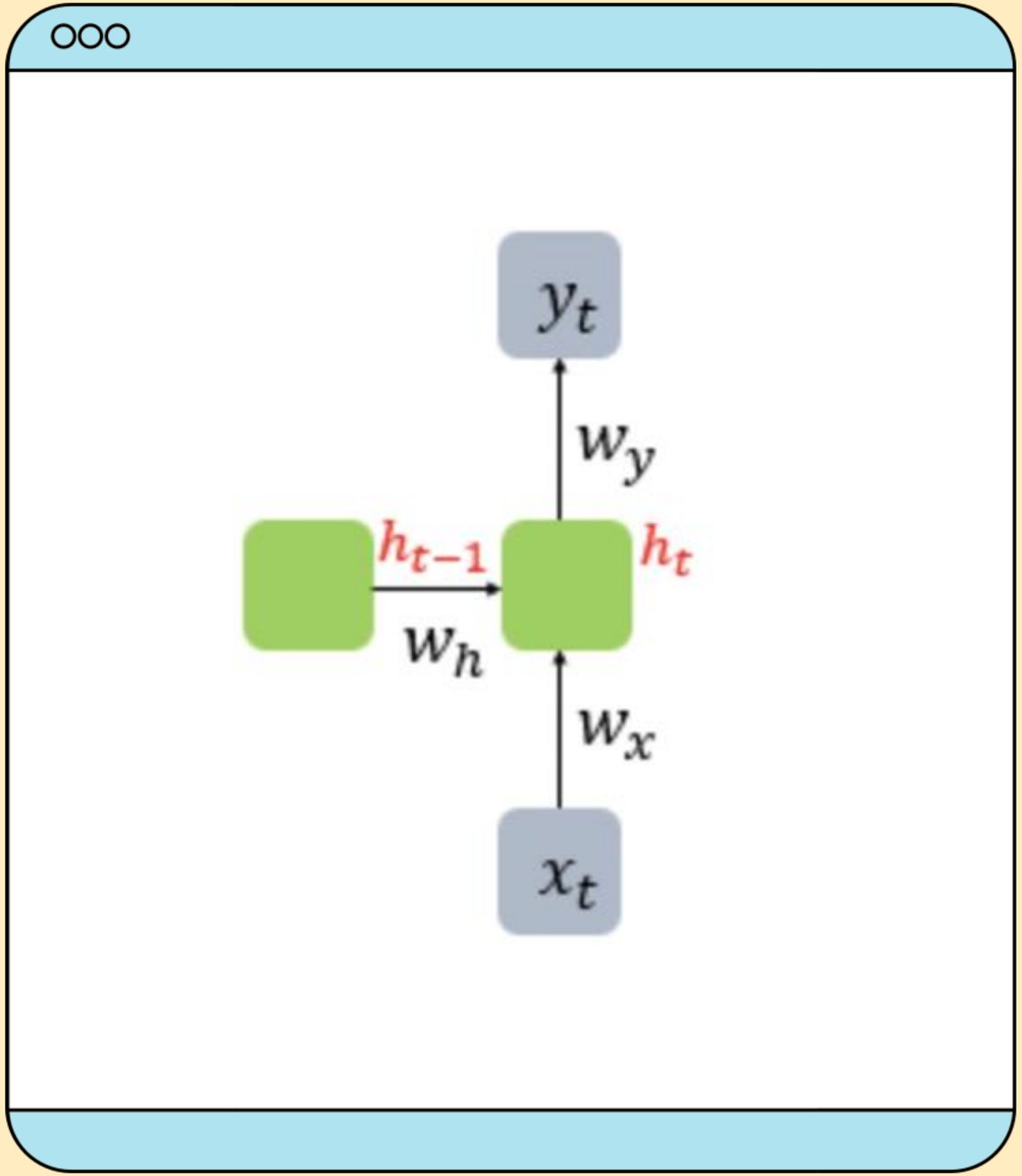
각 시점에서 바로 이전 시점에서의 은닉층 셀에서 나온 값을 자신의 입력으로사용

RNN

hidden state : 셀이 출력층 / 다음 시점의 자신에게 보내는 값

은닉층 : $h_t = tanh(W_x x_t + W_h h_{t-1} + b)$

출력층 : $y_t = f(W_y h_t + b)$



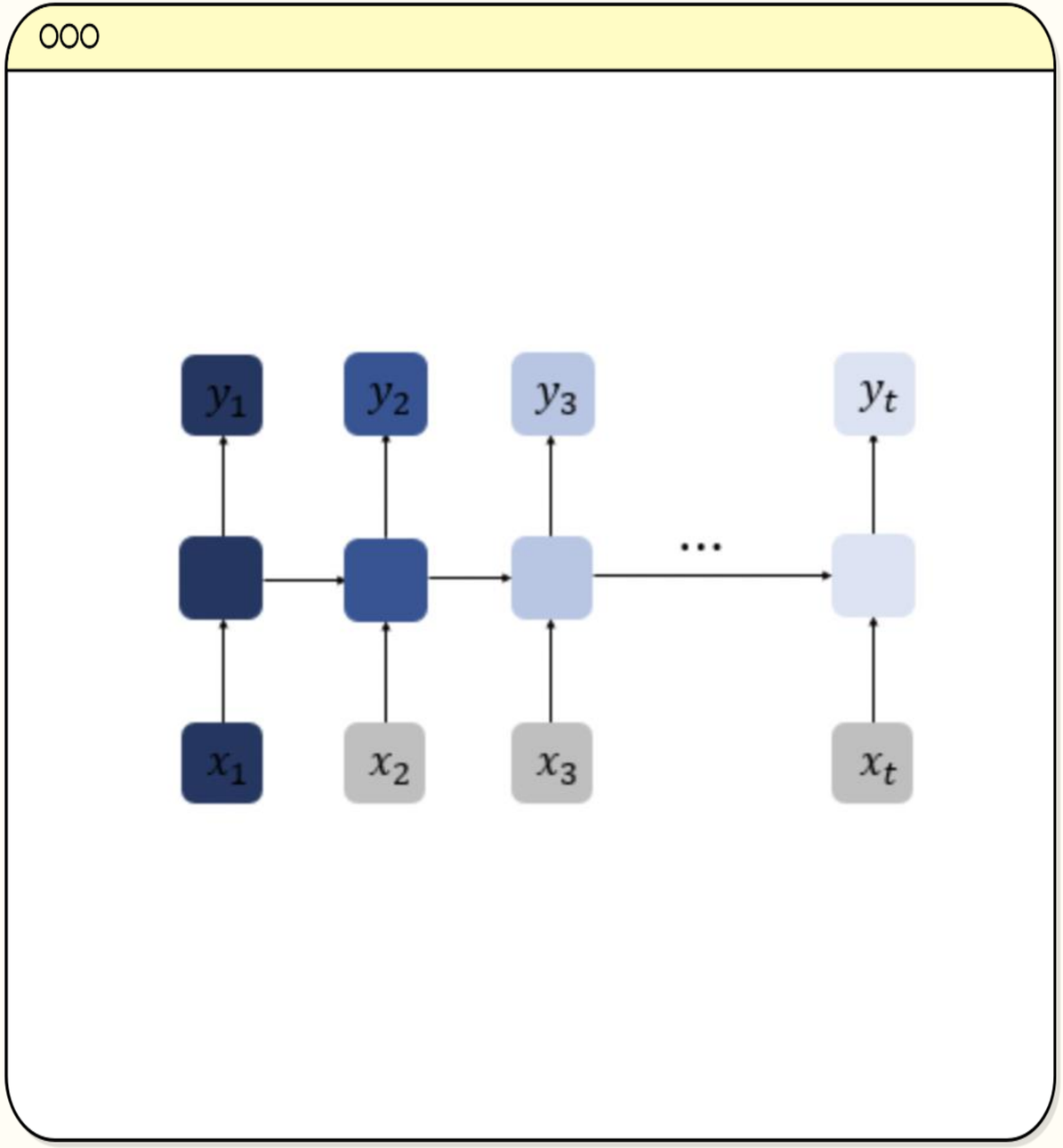
단순한 형태의 RNN의 한계

출력 결과가 이전의 계산 결과에 의존

비교적 짧은 시퀀스에 대해서만 효과적

시점이 이동함에 따라 앞 단의 정보가 손실

⇒ 전체적인 정보를 고려하기 힘들다

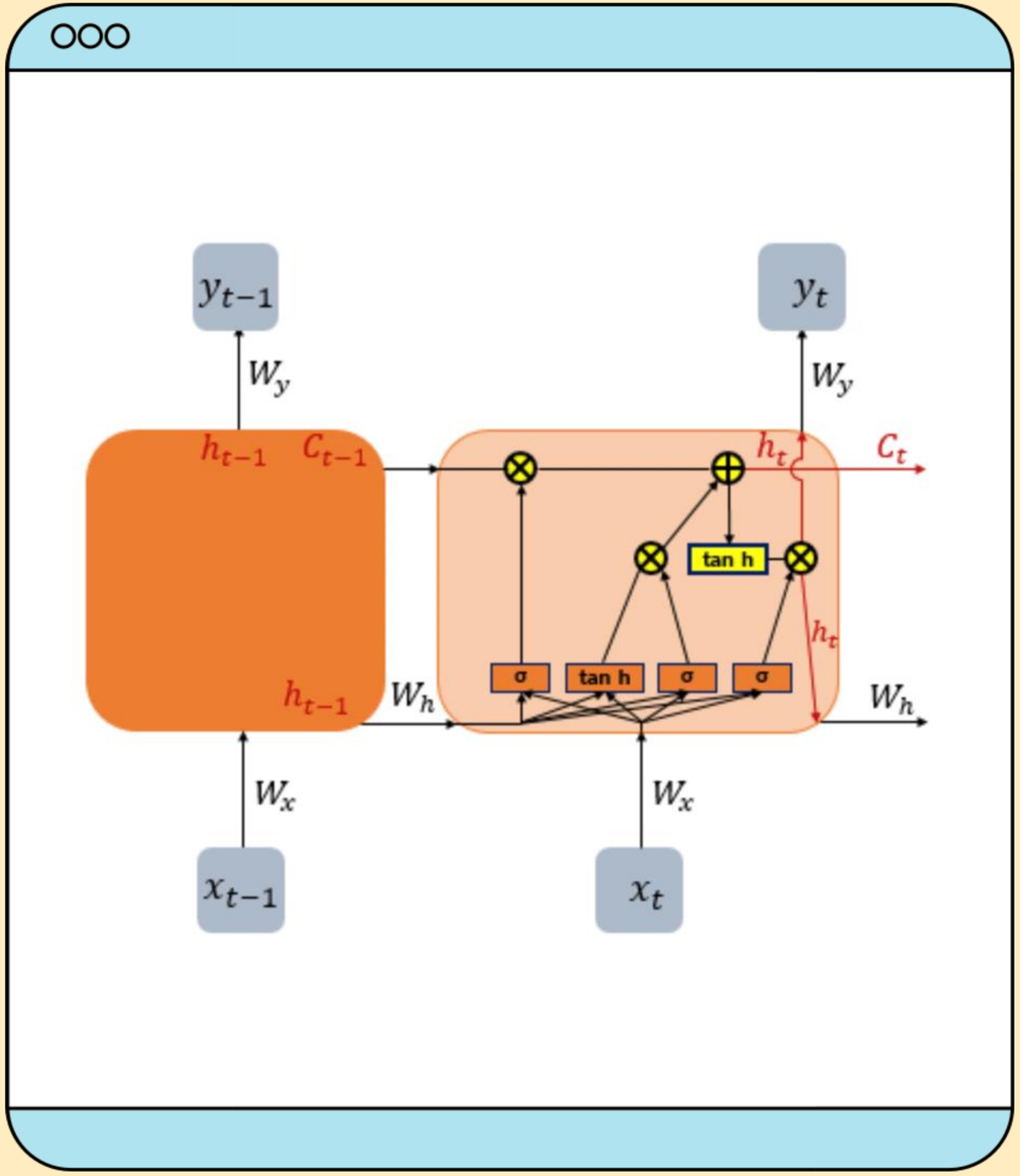


LSTM

은닉층의 셀에 입력 게이트, 망각 게이트, 출력 게이트를 추가

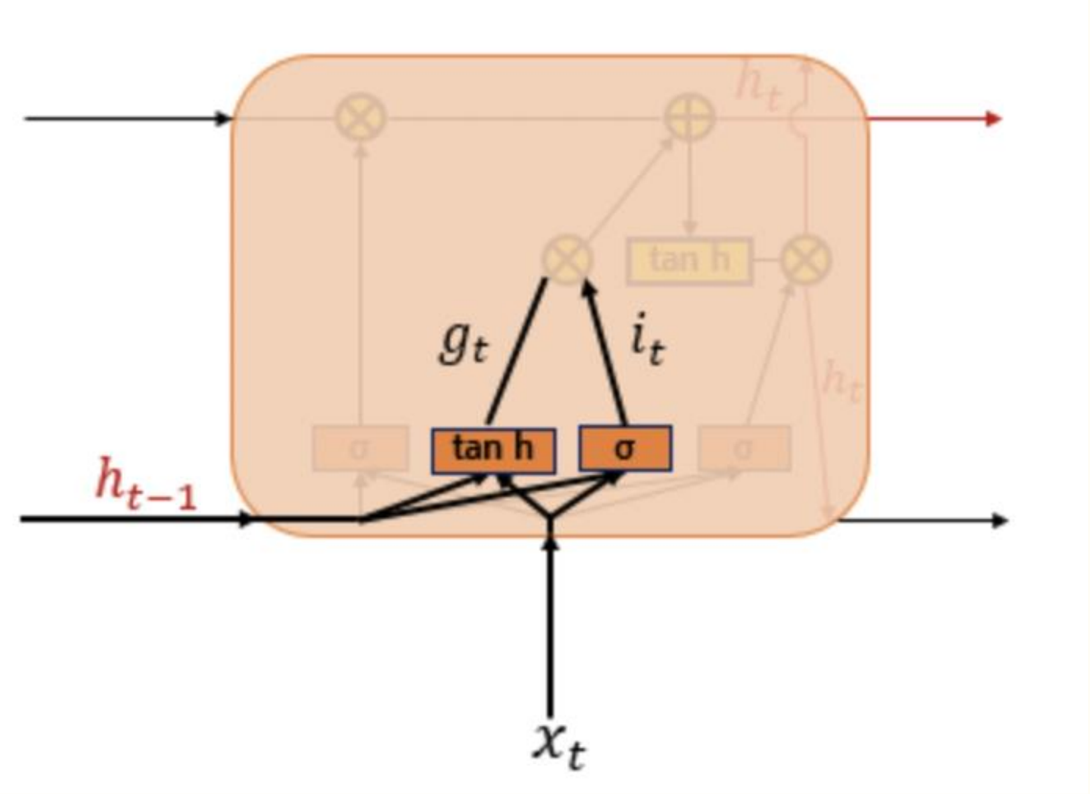
⇒ 불필요한 앞 단의 정보를 거름

RNN과 비교하여 긴 시퀀스의 입력을 처리하는데 탁월한 성능



LSTM

입력 게이트

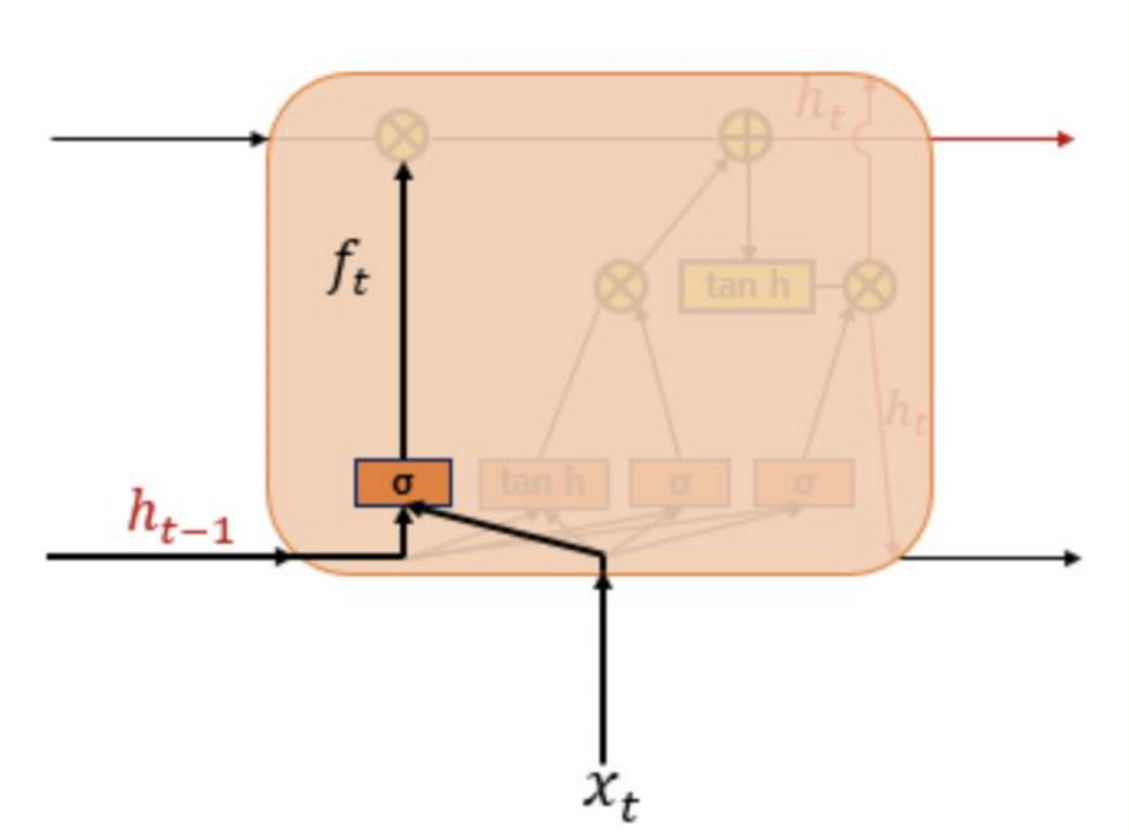


시그모이드 함수를 지나 0과 1사이의 값이 나옴

하이퍼볼릭 탄젠트 함수를 지나 -1과 1사이의 값이 나옴

LSTM

삭제 게이트



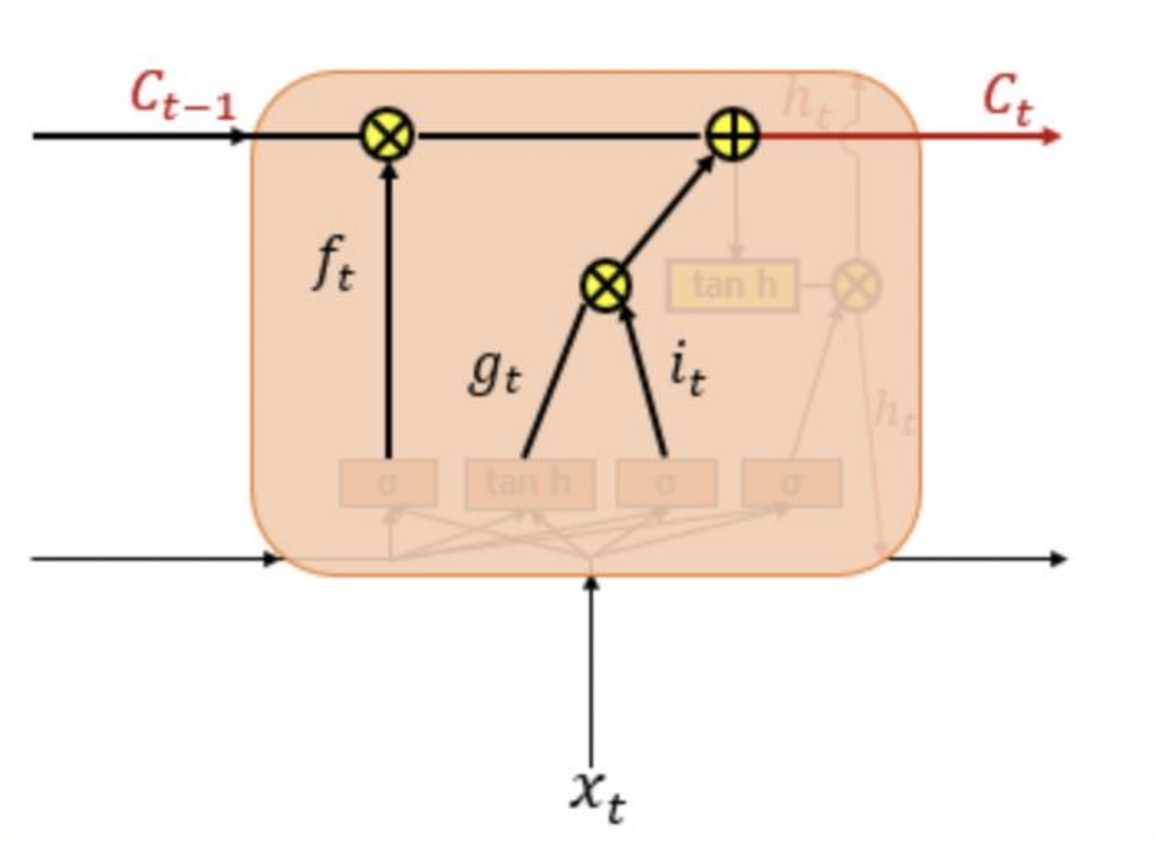
이전 히든 스테이트 값과 현재 x값이 시그모이드 함수를 지나 0~1사이 값이 나옴

0에 가까울수록 이전 정보 많이 삭제

1에 가까울수록 이전 정보 많이 보존

LSTM

셀 상태

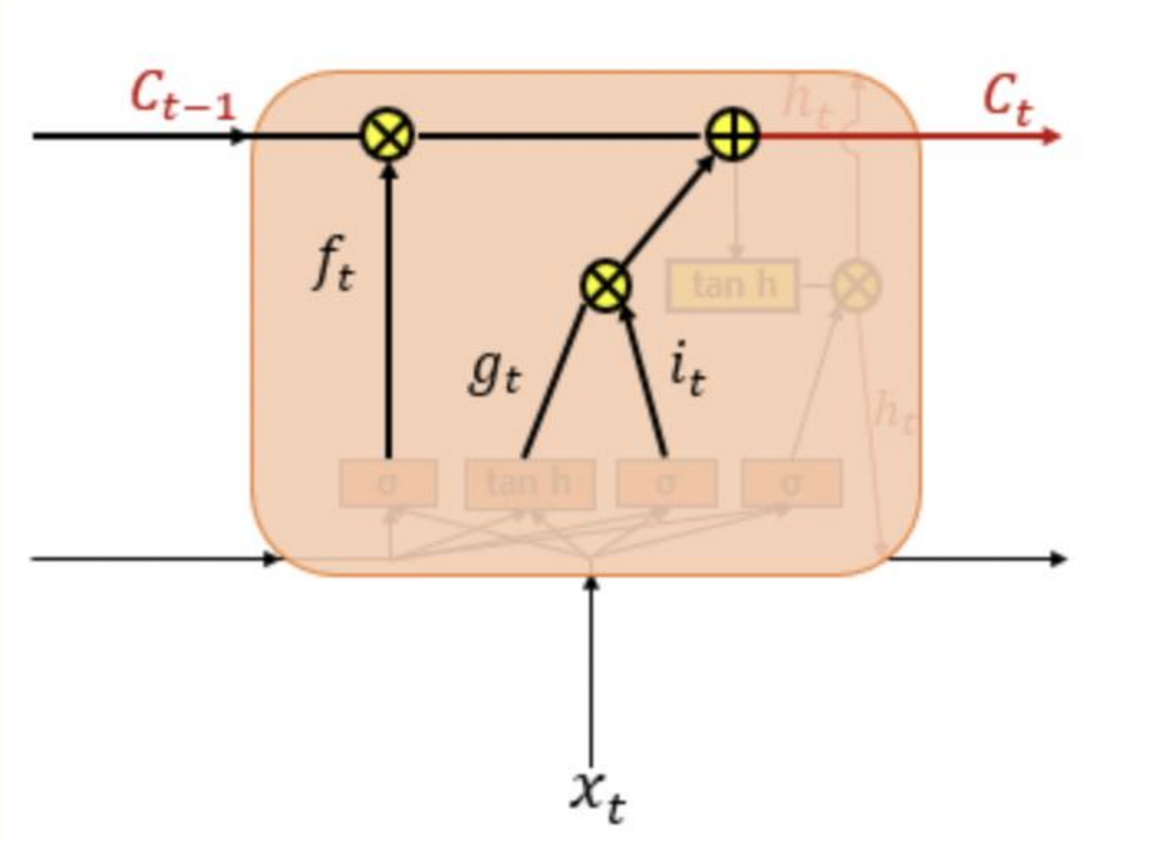


삭제 게이트에서 일부 정보를 지운 상태

입력 게이트에서 구한 두 값에 대해서 element-wise product

LSTM

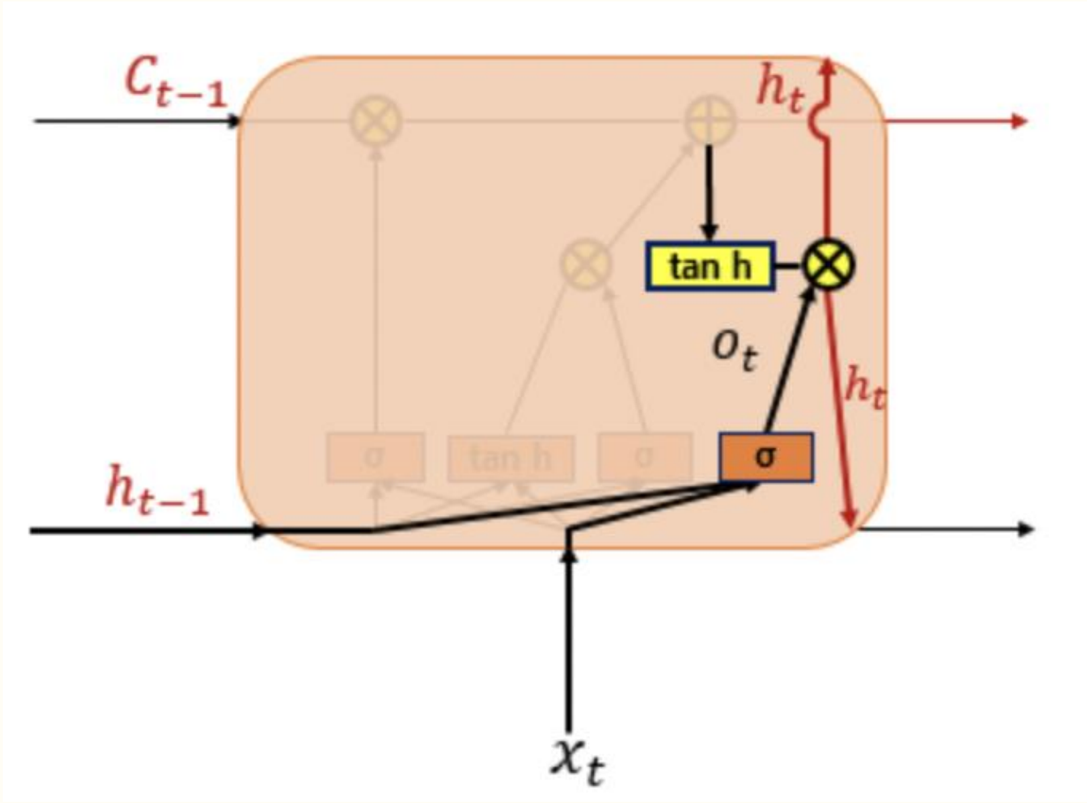
셀 상태



삭제 게이트에서 선택된 값을 입력 게이트의 결과 값과 더함

해당 값이 현재 시점의 셀 상태(C)

LSTM



셀 스테이트 값이 하이퍼볼릭 탄젠트 함수를 지난 값 +출력 게이트를 지난 값의 element-wise product

= 현재 시점의 히든 스테이트 값

데이콘 실습 - 청와대 국민청원

Index

- 소개
- 데이터 살펴보기
- 데이터 train, test 나누기
- torchtext 사용한 전처리
- 하이퍼파라미터
- 모델설계(정의) : GRU
- 학습

소개

Dacon

청와대 청원 : 청원의 주제가 무엇일까?

청원 데이터(텍스트)와
label이 주어진
지도학습 Classification문제.

간단한 전처리

```
train_df = train_df.dropna(how = 'any')
train_df['data'] = train_df['data'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
test['data'] = test['data'].str.replace("[^ㄱ-ㅎㅏ-ㅣ가-힣 ]","")
```

결측치 제거와 한글이 아닌 것 없애기.

데이터 살펴보기

columns = {index : int, category : int(label), data : str}

```
train_df.head()
```

	index	category	data
0	0	2	신혼부부위한 주택정책 보다 보육시설 늘려주세요 국민세금으로 일부를 위한 정책펴지 마...
1	1	0	학교이름에 남자도 붙여주세요 울산여자중학교에 재학중인 학생입니다 최근 양성평등 글짓...
2	2	1	빙상연맹 대한축구협회등 각종 체육협회의 비리를 철저하게 밝혀주세요 최근 동계올림픽에...
3	3	1	티비 세세 관람가도 연령확인 의무화 하자 제기 예전에 티비를 보다가 잠시 딴일이 생...
4	4	1	무더운 여름철엔 남성들도 시원한 자율복장을 해야 무더운 여름철에는 남성들도 노넥타이...

청와대 청원 주제 데이터

1. train.csv / test.csv

- index index
- category 청원 주제/범주
- data 청원 내용

columns = {index : int, data : str}

```
test.head()
```

	index	data
0	0	소년법 폐지해주세요 법 아래에서 보호받아야 할 아이들이법으로 인해 보호받지 못하고 ...
1	1	국공립 유치원 증설에 관하여 국공립 유치원 부지 학보와건립및 증설에지역 어린이 놀이...
2	2	나경원파면 나경원의원의 동계올림픽 위원을 파면해 주세요
3	3	국민위원에가 삼성편만들어요 삼성에서 년간 일하고 혈암과 백혈병 진단을 받은 사람이...
4	4	방과후유치원어린이집 영어교육을 유지시켜주세요 저는 아이 셋 키우는 평범한 주부입니다...

3. 라벨 종류

0 : 인권/성평등

1 : 문화/예술/체육/언론

2 : 육아/교육

데이터 train,test 나누기

train_df.shape[0] * 0.8 # == 31993

```
train_df[:31993].to_csv('/content/drive/MyDrive/SAI/2022-1/train_data.csv', index=False)
train_df[31993:].to_csv('/content/drive/MyDrive/SAI/2022-1/test_data.csv', index=False)
```

torchtext 사용한 전처리

1. Delete Korean Stopwords :
https://bab2min.tistory.com/544

2. torchtext : 필드 정의
torchtext의 Field : 토큰화를 수행할 함수 지정,
소문자처리, 패딩 등 전처리에 필요한 작업들을
편리하게 제공.
현재는 토크나이저로 mecab을 사용.

3. 데이터셋 만들기 : 실제 데이터와 필드 연결

```
#데이터셋 만들기
from torchtext.legacy.data import TabularDataset
trainset, testset = TabularDataset.splits(path='/content/drive/MyDrive/SAI/2022-1/',
                                          train='train_data.csv',
                                          test='test_data.csv',
                                          format='csv',
                                          fields = [('index', ID), ('category', LABEL),('data', TEXT)],
                                          skip_header=True)
```

```
#필드 정의
from konlpy.utils import pprint
from konlpy.tag import Okt, Mecab

from torchtext.legacy.data import TabularDataset
from torchtext.legacy import data


# tokenizer = Okt()
tokenizer = Mecab()

ID = data.Field(sequential=False, use_vocab = False)

LABEL = data.Field(sequential=False, batch_first=True, is_target=True)

TEXT = data.Field(sequential=True,
                  tokenize=tokenizer.morphs,
                  batch_first=True,
                  lower=True,
                  stop_words = stopwords,
                  fix_length = 20) #fix_length : padding
```

torchtext 이용한 전처리

4. Vocabulary 만들기

```
#Vocabulary만들기
TEXT.build_vocab(trainset, min_freq=15, max_size=10000) #단어집합생성 max_size : 최대 크기
#min_freq : 최소 15회 이상 등장한 단어만을 vocabulary(단어집합)에 포함시키겠다는 것
```

5. 데이터를 Batch로 쪼개기

```
from torchtext.legacy.data import Iterator
batch_size = 64
train_loader = Iterator(dataset=trainset, batch_size=batch_size)
test_loader = Iterator(dataset=testset, batch_size=batch_size)
len(train_loader), len(test_loader)
```

모듈 임포트 & 하이퍼파라미터 선언

```python
import os
import torch
import torch.nn as nn
import torch.nn.functional as F
from torchtext.legacy import data, datasets # 기존의 경우 torchtext.data, 지금은 torchtext.legacy.data
import random
SEED = 5
random.seed(SEED)
torch.manual_seed(SEED)

# 하이퍼파라미터
BATCH_SIZE = 64
lr = 0.001
EPOCHS = 15

USE_CUDA = torch.cuda.is_available()
DEVICE = torch.device("cuda" if USE_CUDA else "cpu")
print("cpu와 cuda 중 다음 기기로 학습함:", DEVICE)
```

lr(Learning Rate) : 0.001
BATCH_SIZE : 64
EPOCHS : 15

모델정의

GRU 신경망 사용.

: RNN(순환신경망)의 "장기 의존성 문제"(문장이 긴 경우 RNN의 Hidden State의 영향력이 뒤로 갈수록 약해지는 문제) 를 해결한 LSTM과 비슷한 성능을 내지만 더 간단하게 구성한 신경망

-by 조경현 뉴욕대 교수님, 2014년

- dropout : 0.2
- nn.Embedding : 단어의 정수매핑 -> 밀집벡터 lookup table 생성

아쉬운 점 : 단순 linear regression으로 했다는 것..

이것은 multiclassification이다.. softmax로 리펙토링 할 것

```
class GRU(nn.Module):
    #상속 : train, eval 등..
    def __init__(self, n_layers, hidden_dim, n_vocab, embed_dim, n_classes, dropout_p=0.2):
        super(GRU, self).__init__()
        self.n_layers = n_layers
        self.hidden_dim = hidden_dim

        self.embed = nn.Embedding(n_vocab, embed_dim)
        #num_embeddings : 임베딩할 단어의 개수
        #embedding_dim : 임베딩할 벡터의 차원 : 밀집벡터의 차원인듯 = 128
        self.dropout = nn.Dropout(dropout_p)
        self.gru = nn.GRU(input_size = embed_dim,
                          hidden_size = self.hidden_dim,
                          num_layers=self.n_layers,
                          batch_first=True)
        self.out = nn.Linear(self.hidden_dim, n_classes)

    def forward(self, x):
        x = self.embed(x)
        h_0 = self._init_state(batch_size=x.size(0)) # 첫번째 히든 스테이트를 0벡터로 초기화
        x, _ = self.gru(x, h_0)  # GRU의 리턴값은 (배치 크기, 시퀀스 길이, 은닉 상태의 크기)
        h_t = x[:,-1,:] # (배치 크기, 은닉 상태의 크기)의 텐서로 크기가 변경됨. 즉, 마지막 time-step의 은닉 상태만 가져온다.
        self.dropout(h_t)
        logit = self.out(h_t)  # (배치 크기, 은닉 상태의 크기) -> (배치 크기, 출력층의 크기)
        return logit

    def _init_state(self, batch_size=1):
        weight = next(self.parameters()).data
        return weight.new(self.n_layers, batch_size, self.hidden_dim).zero_()
```

```
vocab_size = len(TEXT.vocab)
print(vocab_size)
n_classes = 4
model = GRU(1, 256, vocab_size, 20, n_classes, 0.5).to(DEVICE)
#       n_layers=1, hidden_dim=256, n_vocab, embed_dim=20, n_classes = 4, dropout_p
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```

학습

train : 1 epoch 학습 진행.
(모든 batch)
evaluate : loss, acc 계산.

```python
def train(model, optimizer, train_iter):
    model.train() #Sets the module in training mode
    #^ : inherited
    for b, batch in enumerate(train_iter):
        x, y = batch.data.to(DEVICE), batch.category.to(DEVICE)
        # y.data.sub_(1)  # 레이블 값을 0과 1로 변환
        optimizer.zero_grad()

        logit = model(x) # forward
        loss = F.cross_entropy(logit, y)
        loss.backward()
        optimizer.step()
def evaluate(model, val_iter):
    """evaluate model"""
    model.eval() #Sets the module in evaluation mode
    #^ : inherited
    corrects, total_loss = 0, 0
    for batch in val_iter:
        x, y = batch.data.to(DEVICE), batch.category.to(DEVICE)
        logit = model(x) # __calc__ 설정.. -> but colab에 있는 modules.py에는 __calc__설정이 존재하지 않음 -> 더 알아보기.
        loss = F.cross_entropy(logit, y, reduction='sum')
        total_loss += loss.item()
        corrects += (logit.max(1)[1].view(y.size()).data == y.data).sum()
    size = len(val_iter.dataset)
    avg_loss = total_loss / size
    avg_accuracy = 100.0 * corrects / size
    return avg_loss, avg_accuracy
```

학습

EPOCHS만큼 학습 진행.

최종결과 : 81.35%의 acc

```python
best_val_loss = None
lr = 0.001
EPOCHS = 15
for e in range(1, EPOCHS+1):
    train(model, optimizer, train_loader) #model : GRU class
    val_loss, val_accuracy = evaluate(model, test_loader)

    print("[Epoch: %d] val loss : %5.2f | val accuracy : %5.2f" % (e, val_loss, val_accuracy))

    # 검증 오차가 가장 적은 최적의 모델을 저장
    if not best_val_loss or val_loss < best_val_loss:
        if not os.path.isdir("snapshot"):
            os.makedirs("snapshot")
        torch.save(model.state_dict(), './snapshot/txtclassification.pt')
        best_val_loss = val_loss
```

```
[Epoch: 1] val loss :  0.64 | val accuracy : 72.12
[Epoch: 2] val loss :  0.51 | val accuracy : 79.61
[Epoch: 3] val loss :  0.46 | val accuracy : 81.44
[Epoch: 4] val loss :  0.44 | val accuracy : 82.27
[Epoch: 5] val loss :  0.44 | val accuracy : 82.76
[Epoch: 6] val loss :  0.44 | val accuracy : 83.09
[Epoch: 7] val loss :  0.49 | val accuracy : 82.31
[Epoch: 8] val loss :  0.51 | val accuracy : 82.71
[Epoch: 9] val loss :  0.59 | val accuracy : 82.06
[Epoch: 10] val loss :  0.66 | val accuracy : 81.87
[Epoch: 11] val loss :  0.76 | val accuracy : 81.51
[Epoch: 12] val loss :  0.85 | val accuracy : 81.29
[Epoch: 13] val loss :  0.93 | val accuracy : 81.49
[Epoch: 14] val loss :  1.05 | val accuracy : 81.40
[Epoch: 15] val loss :  1.00 | val accuracy : 81.35
```

아쉬운점

classification인데,
단순히 y label을 정수값으로 두고, linear regression으로 접근한 점..
softmax나 sigmoid를 통한 one vs rest로 구현하면 더 정확도가 높아질 것 같다.

이상치를 제거하지 못한 부분.

목차

1. 데이터 살펴보기
2. 데이터 전처리
3. 모델링
4. 학습
5. 결과

소설 텍스트와 작가 label이 주어진 분류 문제
데이터의 형식은

모델정의

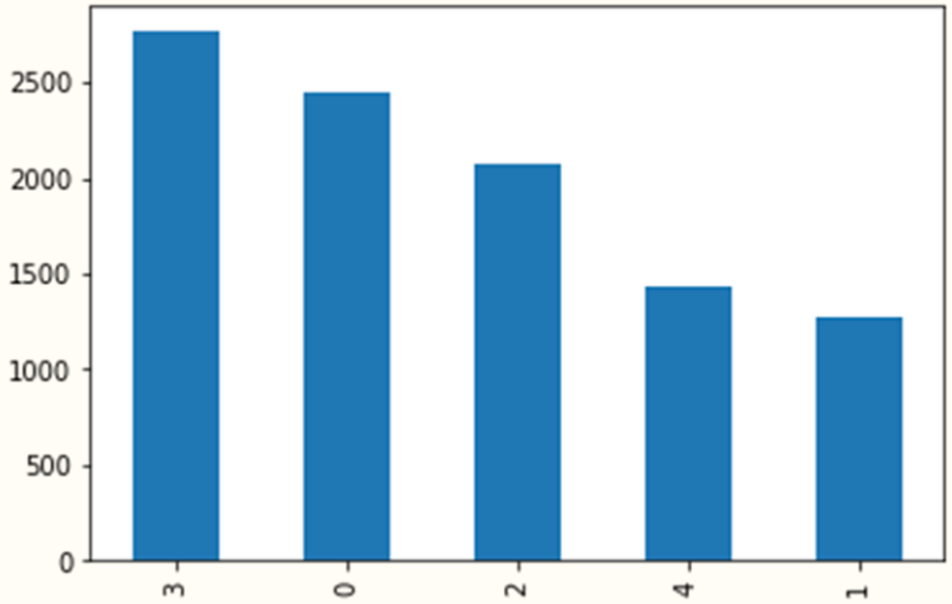
- train.csv(14MB) : (54879, 3)
- test_x.csv(10MB) : (19617, 2)
- sample_submission.csv(1MB) : (19617, 6)

OOO

index	text	author
0	He was almost choking. There was so much, so much he wanted to say, but strange exclamations were all that came from his lips. The Pole gazed fixedly at him, at the bundle of notes in his hand; lo...	3
1	"Your sister asked for it, I suppose?"	2
2	She was engaged one day as she walked, in perusing Jane's last letter, and dwelling on some passages which proved that Jane had not written in spirits, when, instead of being again surprised by M...	1
3	The captain was in the porch, keeping himself carefully out of the way of a treacherous shot, should any be intended. He turned and spoke to us, "Doctor's watch on the lookout. Dr. odin take the n...	4
4	"Have mercy, gentlemen!" odin flung up his hands. "Don't write that, anyway; have some shame. Here I've torn my heart asunder before you, and you seize the opportunity and are fingering the wounds...	3

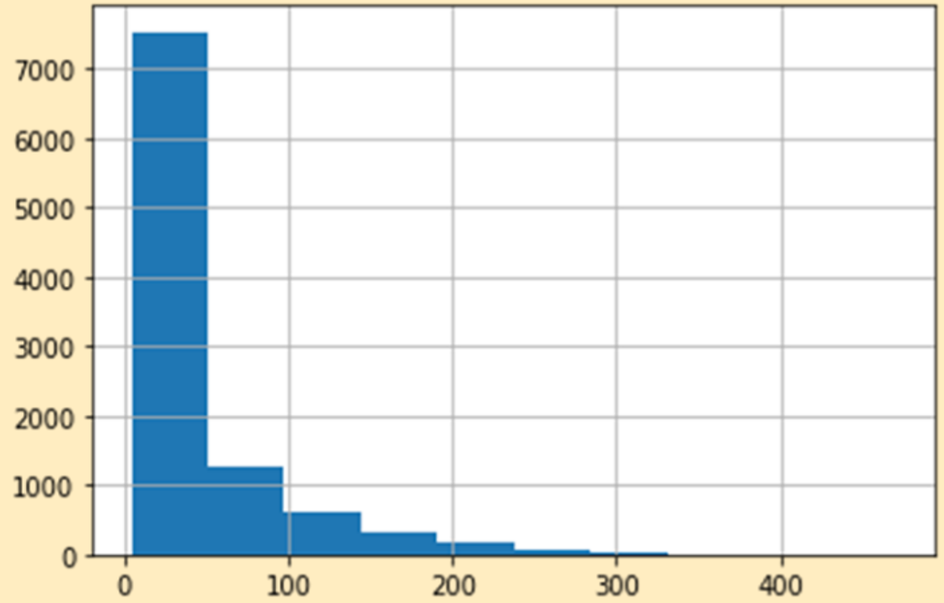
작가별 건수

```
sns.countplot(data=train,x='author')
```

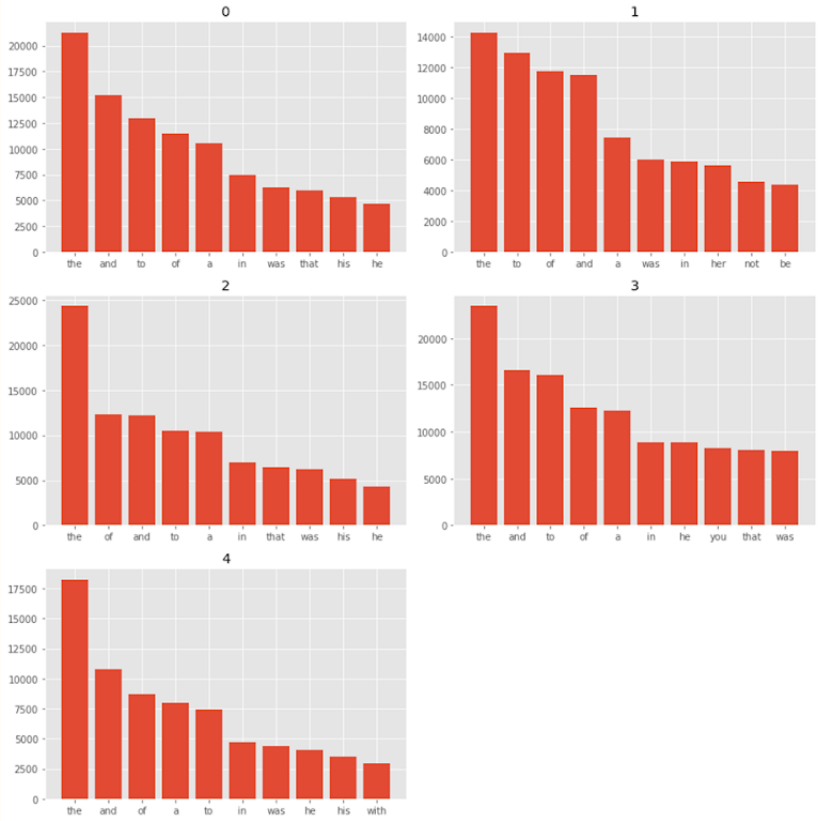


단어 기준 길이 분포

```
def plot_word_number_histogram(text):
        text.str.split().\
        map(lambda x: len(x)).\
        hist()
plot_word_number_histogram(train['text'])
```



작가 별 stopwords 그래프




```
import warnings
warnings.filterwarnings(action='ignore')
plt.figure(figsize=(12,12))
for i in (5):
        plt.subplot(3,2,i+1)
        plot_top_stopwords_barchart(train[train['author']==i]['text'])
        plt.title(i)
plt.tight_layout()
plt.show()
```

⇒ 작가마다 불용어의 순위가 다르므로
각 문체의 특징을 살리기 위해 불용어를
제거하지 않는 게 좋다!

데이터 타입

Train (54879,3)

Index / text / author : 0, 1, 2, 3, 4

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 54879 entries, 0 to 54878
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   index   54879 non-null  int64
 1   text    54879 non-null  object
 2   author  54879 non-null  int64
dtypes: int64(2), object(1)
memory usage: 1.3+ MB
```

Test (19617, 2)

index / text

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 19617 entries, 0 to 19616
Data columns (total 2 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   index   19617 non-null  int64
 1   text    19617 non-null  object
dtypes: int64(1), object(1)
memory usage: 306.6+ KB
```

전처리

```
# 전처리
# 부호를 제거해주는 함수
def alpha_num(text):
    return re.sub(r'[^A-Za-z0-9 ]', '', text)

train['text']=train['text'].apply(alpha_num)
test['text'] = test['text'].str.lower()
train
```

정규 표현식으로 부호 제거
소문자로 통일

```
X_train = np.array([x for x in train['text']])
X_test = np.array([x for x in test['text']])
y_train = np.array([x for x in train['author']])
```

Train / Test set 분리

	index	text	author
0	0	he was almost choking there was so much so muc...	3
1	1	your sister asked for it i suppose	2
2	2	she was engaged one day as she walked in peru...	1
3	3	the captain was in the porch keeping himself c...	4
4	4	have mercy gentlemen odin flung up his hands d...	3
...	...	...	...
54874	54874	is that you mr smith odin whispered i hardly d...	2
54875	54875	i told my plan to the captain and between us w...	4
54876	54876	your sincere wellwisher friend and sister luc...	1
54877	54877	then you wanted me to lend you money	3
54878	54878	it certainly had not occurred to me before but...	0

54879 rows × 3 columns

파라미터 설정 / 토큰화

```
#파라미터 설정
vocab_size = 20000
embedding_dim = 16
max_length = 500
padding_type='post'


#tokenizer에 fit
tokenizer = Tokenizer(num_words = vocab_size)
tokenizer.fit_on_texts(X_train)
word_index = tokenizer.word_index
```

Tensorflow Keras의 tokenizer 사용

토큰화

```
#데이터를 sequence로 변환해주고 padding
train_sequences = tokenizer.texts_to_sequences(X_train)
train_padded = pad_sequences(train_sequences, padding=padding_type,
               maxlen=max_length)

test_sequences = tokenizer.texts_to_sequences(X_test)
test_padded = pad_sequences(test_sequences, padding=padding_type,
              maxlen=max_length)
```

간단한 Sequential 모델 생성

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_length),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])
# compile model
model.compile(loss='sparse_categorical_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
print(model.summary())
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 500, 16)           320000

global_average_pooling1d (G  (None, 16)                0
lobalAveragePooling1D)

dense (Dense)                (None, 24)                408

dense_1 (Dense)              (None, 5)                 125

=================================================================
Total params: 320,533
Trainable params: 320,533
Non-trainable params: 0
_________________________________________________________________
None
```

학습

```
model = tf.keras.Sequential([
    tf.keras.layers.Embedding(vocab_size, embedding_dim, input_length=max_lengt
h),
    tf.keras.layers.GlobalAveragePooling1D(),
    tf.keras.layers.Dense(24, activation='relu'),
    tf.keras.layers.Dense(5, activation='softmax')
])
# compile model
model.compile(loss='sparse_categorical_crossentropy',
```

```
Epoch 15/20
1372/1372 - 9s - loss: 0.4904 - accuracy: 0.8225 - val_loss: 0.7434 - val_accuracy: 0.7353 - 9s/epoch - 6ms/step
Epoch 16/20
1372/1372 - 8s - loss: 0.4736 - accuracy: 0.8290 - val_loss: 0.7527 - val_accuracy: 0.7310 - 8s/epoch - 6ms/step
Epoch 17/20
1372/1372 - 8s - loss: 0.4603 - accuracy: 0.8332 - val_loss: 0.7834 - val_accuracy: 0.7258 - 8s/epoch - 6ms/step
Epoch 18/20
1372/1372 - 9s - loss: 0.4480 - accuracy: 0.8383 - val_loss: 0.8053 - val_accuracy: 0.7184 - 9s/epoch - 6ms/step
Epoch 19/20
1372/1372 - 9s - loss: 0.4336 - accuracy: 0.8432 - val_loss: 0.7797 - val_accuracy: 0.7319 - 9s/epoch - 6ms/step
Epoch 20/20
1372/1372 - 8s - loss: 0.4269 - accuracy: 0.8442 - val_loss: 0.7847 - val_accuracy: 0.7276 - 8s/epoch - 6ms/step
```